

VERİ YAPILARI

Tanım: Bilgisayarda kullanılan değişkenler birer veridir. Bu verilerin yönetilmesi anlamlıdır. Standart kullanılan değişken tipleri her ne kadar birer veri yapısı olsa da daha karmaşık veriler, farklı tip yapı gerektir.

Yeryüzündeki bilgilerin toplanması veri yapısı ve yönetiminin değerli kılardır.

Veri Yapıları

Primitive (İlkel)

- ↳ int
- ↳ float
- ↳ String
- ↳ char
- ↳ pointer

Değişken tipleri

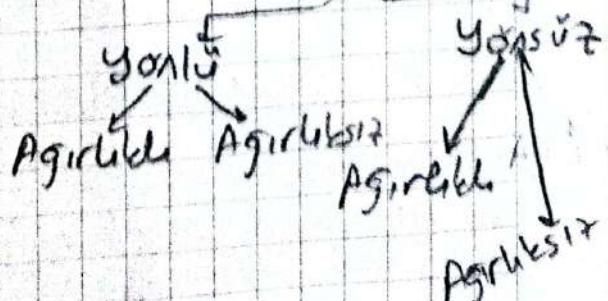
Non-Primitive (Gelişmiş)

- ↳ Linear
 - ↳ Diziler
 - ↳ Stack (Yığın)
 - ↳ Queue
 - ↳ Lists

- ↳ Non-Linear
 - ↳ Tree
 - ↳ Binary Tree
 - ↳ Binary Search T
 - ↳ B-tree
 - ↳ B+ tree
 - ↳ AVL
 - ↳ Red-black
 - ↳ 3-5 tree

Temelidir ***

Graphs (Güçge)



NOT1) Graph'lar bu işin temelidir.

2) Argaşlar: Veriler çok büyük olduğunda bunları tutulmasına ve yönetimine yardım eder

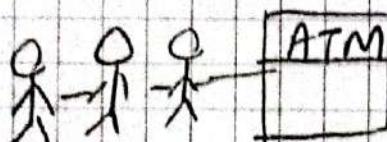
3) Listeler: Çeşitli formatlarda (tele yönlü - çift yönlü) olmak üzere verilerin tutulmasına ve yönetimine yardım eder.

4) Stack (Yığın): LIFO → Last In First Out



"Son giren ilk çıkar" montajına göre verileri tutar, yönetir.

5) Kuyruk: ATM'den sıraya girip para çekmeye çalışan insanların modelidir.



FIFO (First in first out)

"İlk giren ilk çıkar" montajına göre çalışır.

6) Diziler: Aynı tipdeki birbirinden farklı veriyi tutan yapılardır.

Temel İşlemler: Stack, Queue, Lists, Tree

1) Oluşturma

2) Arama (Searching)

3) Silme (Deletion)

4) Gezinme (Traversal) ***

5) Updation (Güncelleme)

NOT: Data \Rightarrow Gogul (Veriler)
Dartum \Rightarrow Telci (Veri)

DiziLER

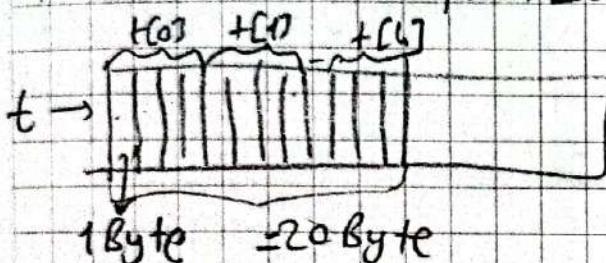
Aynı tipteki verilerin tutulduğu yapılardır.

Avantajları

+ toplamaması kolaydır.

`int t[] = new int[5];`

`float c[][] = new float[3][5];`



`t[0] = 10;`
_____ 20 bit 4 bit
0 --- 0001010
32 bit

byte `x[] = new byte[5];`
8 bit 00001010
word: 4 bit

+ Dogruden erisimi destekler

Desavantajları

- Gereksiz yer işgaline sebep olur.

(Fırmantasyon sebep olur.)

\hookrightarrow yer olduğu halde kullanılmama durumu.

\rightarrow Dizilerin veri yapılarda çok kullanılmaz.

- Dizinin boyutu, eleman sayısı sonradan genişletilemez, artırlamaz.

- Dizilerin dinamik olmamaları

Dizi istenleri

1) Oluşturma

2) Arama işlemi `int f[] = new int[10];`

3) Updation (-) `(f[3], f[5] = -1)`

~~X~~ `a[0] a[1] a[2] a[3]`
soy 10 20 30 0040

Ödev: 5 elementli bir dizideki 1. elementi silen bilgisayar kodunu yazınız.

h) Silme (+)

$$a[4] = \{10, 20, 30, 40\}$$

$a[0]$ $a[1]$ $a[2]$ $a[3]$
10 20 30 40

$a[0]$ $a[1]$ $a[2]$
20 30 40

Dizinin 1. elementini silme işlemi

10	20	30	40
20	30	40	

Ara'dan Silme işlemi

10 20 ~~30~~ 40

10	20	40
10	20	40

Sondan Silme

10 20 30 40

10	20	30	40

Silme
görmeme

NOT: dizilerdeki silme işlemleri için sayısal kullanım teritiğidir. Özetlikle bir den fazla işlem yapılan dizilerde işlem sayısının bilinmesi gereklidir.

Veri Yapılarında Class Kullanımı

Tanım: Dinamik bir şekilde çalışma gerekligi veri yapılarında classlar kullanılır.

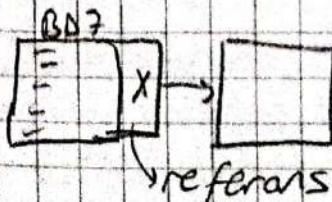
1) Class'la aynı isimli metod constructor'dır
class sınıf {

~~public~~ sınıf() { Sout ("constructor"); }
}

2) Class ismiyle tanımlanan değişken referansdır
class sınıf {

~~sınıf x;~~ → Referans
}

 BD7 = new sınıf();
 BD7.x = new sınıf();



class Araba {

 Araba next;

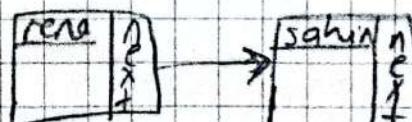
 renzo = new Araba();

 renzo.next = null;

 sahin = new Araba();

 sahin.next = null;

 renzo.next = sahin;



Ogrenci
Ali Veli Osman

class Ogrenci {

 Ogrenci el;

 Ogrenci Ali = new Ogrenci();

 Ali.el = null;

 Ogrenci Osman = new Ogrenci();

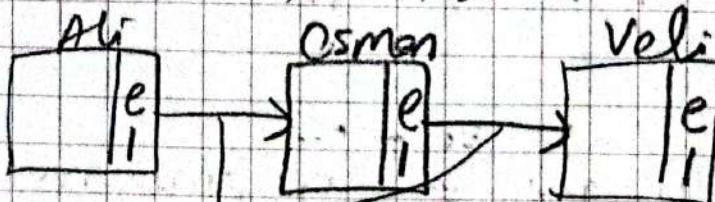
 Osman.el = null;

 Ogrenci Veli = new Ogrenci();

 Veli.el = null;

 Ali.el = osman;

 Ali.el.el = Veli; // Osman.el = Veli.;



Liste

Referans Bağlantıları: sağlar

Algoritmalar Analizi (Basit Sekli)

Tanım: Bir yazılım, program geliştirirken algoritma da geliştirilir.

Geliştirilen algoritmanın doğru çalışması gereklidir. Doğru çalışma algoritmanın ne kadar iyi olduğu meselesini gözmek için analiz edilmesi gereklidir. Bu analiz işlemi dilten bağımsız bir şekilde yapılabilir.

Analiz işlemiinde iki noktaya dikkat edilir:

1. Hiz \rightarrow Big O Notasyonu \rightarrow zaman karmaşıklığı
2. İşgal ettiği hafıza boyutudur. \rightarrow Uzay karmaşıklığı

Analizde 3 türlü yapı mercuttur:

- 1) Big Oh $\Rightarrow \boxed{O(n)}$ \Rightarrow Paul Bachman tarafından tanımlanmıştır
* en adekkileren
 \rightarrow zaman karmaşıklığını gösterir.
✓ zaman karmaşıklığının en üst sınırını gösterir.
* en kötü durum analizini gösterir.
* Yapılan işlem elde edilecek $T(n)$ fonksiyonunu ifade etmektedir.
(Basittir, aksıdan önemlidir.)
- 2) Big Ω Notasyonu \Rightarrow zaman karmaşıklığında alt sınırı gösterir.
* En iyi durum analizi dir.
* Bir algoritmanın çalışabileceği zaman karmaşıklığı aksından en ideal durumunu gösterir.

$n \rightarrow$ ne kadar

3) Big Oh (Teta) Notasyonu

\rightarrow Big Oh Notasyonu $O(n)$ ile $\Omega(n)$ notasyonu arasındaki ortaklama durum analizi dir.

$T(n)$ Hesabi: Bir algoritmanın kullanılen değişken / değişkenleri göz önüne alarak ne kadarlık çalışma zamanını n açısından ortaya konmasıdır.

* Değişken atamaları 1 birimdir.

```
public static int topla (int[] A){  
    int topla = 0; → 1  
    for( int i=0; i < N; i++) { → N+1  
        topla += A[i]; } → N  
    return topla; → 1 birim zaman  
}
```

$$1 + N + 1 + N + 1$$

$$T(N) = 2N + 3 \Rightarrow \text{Fonksiyon} \Rightarrow O(n)$$

→ Dizideki eleman sayısı

* Döngüler döngü sayısına ile ilişkilidir.

* Kodun ^{sadece} çalışma kısmını analiz ediyoruz.

NOT: Algoritmanın veris mali için gerekli
dan gereklilik teknolojik temel hesaplamaları.

Dizideki Bir Elemanın Aranması:

```
public static int Arama (int[] A, n, int sayı) {  
    int i = 0; → 1  
    while (i < N) { → L + 1  
        if (A[i] == sayı) break; → L  
        i++; → L  
    } → 1  
    if (i < N) return 1; → 1  
    else return -1; → 1  
}
```

$1 + L + 1 + L + L + 1 + 1$

$$T(L) = 3L + 4$$

$$L = n \text{ için}; T(n) = 3n + 4$$
$$O(n) \Rightarrow O(n)$$

İç içe Döngü

```
for (int i = 1; i <= N; i++) {  
    for (int j = 1; j <= M; j++) {  
    }
```

$$\sum_{i=1}^N \sum_{j=1}^M \Rightarrow (N+1)(M+1) \Rightarrow (N+1)(N+1)$$

$$N=M \text{ için} = N^2 + 2N + 1$$

$$T(N) = N^2 + 2N + 1$$

* $O(n)$ 'e dönüştürürken $\Rightarrow O(N^2)$
en büyük işse sahip
demi alıyoruz.

NOT: Değişkenin int'leri farklı olsa bile $T(n)$ hesabında değişken eşitlenmesi yapılır ve tek değişkenle başlı bir fonksiyon elde edilir.

Matris Çarpımı

$N+1 \leftarrow \text{for } (\text{int } i=0; i < N; i++) \{$

$N+1 \leftarrow \text{for } (\text{int } j=0; j < N; j++) \{$

$1 \leftarrow C[i][j] = 0;$

$N+1 \leftarrow \text{for } (\text{int } k=0; k < N; k++) \{$

$C[i][j] += A[i][k] \cdot B[k][j]$

333

$$\sum_{i=0}^N \sum_{j=0}^N \left(1 + \sum_{k=0}^N \right) \Rightarrow N^3 + N^2 \Rightarrow T(N) = N^3 + N^2$$

$$O(N) = N^3$$

Space (Uzay, yer) karmaşıklığı

Uzay karmaşıklığı özellikle değişken boyutu, sayısı çok büyük olduğunda önemlidir.

Değişkenin tipinin seçilmemesi de bu durumla kritiktir.

Örneğin n elemanlı bir tam sayı listesi Java'da kullanılacaksa bunun alan maliyeti $4n$ 'dır.

$\text{int } A = \text{new int}[100];$ (4x100) $\text{int } \text{in, hafızadaki yer}$

$\text{double } B = \text{new double}[40];$ (8x40) $\text{double} \rightarrow 8n$

$\text{double toplam} = 0.0;$ (8)

$= 728 \text{ byte}$

$\begin{cases} \dots \\ \dots \\ \dots \end{cases} \begin{cases} \text{hesap işi, toplan} \\ \text{karmaşıklığı} \end{cases}$

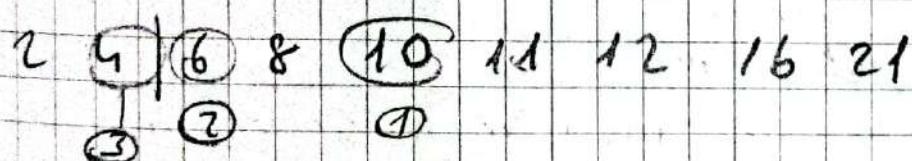
NOT: yer karmaşıklığı hesaplanırken hafızadaki isgal etmesi olduğu alan hesaplanır.
Özellikle büyük dataların olduğu programlar da alan maliyetini ifade eden yapılar asimptotik yapılardır.

NOT 1: Küçük arama algoritması $\log N$ de çalışır.
Logaritmik denince genelde log tabanı düşünülür.
Bunun nedeni $\log 1/0$ mantığıdır.

NOT 2: Recursive fonksiyonlar, böl ve yönet algoritmaları genelde $\log N$ de çalışır.

NOT 3: Logaritmik ifadelerin kullanımı sayısal verilerin görüntülenmesinde / ifade etmesinde yarar sağlar.

İlk Arama



$$\log_2 N \quad N = 1024$$

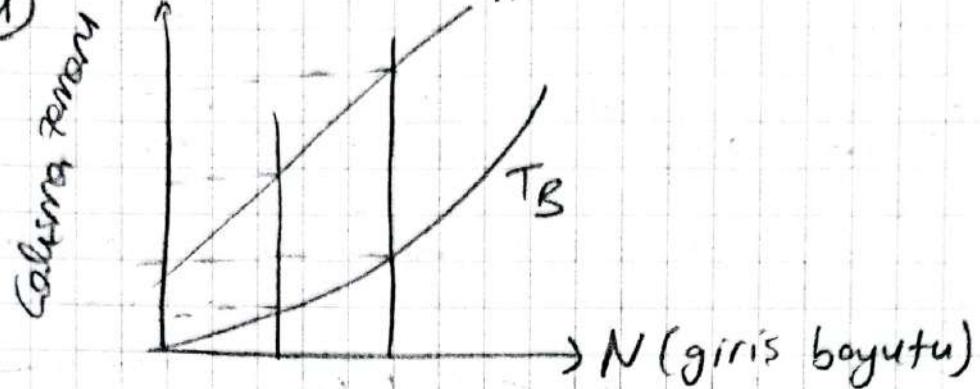
$$\log_2 1024 = 10$$

Linear Search 10at

Binary Search 10

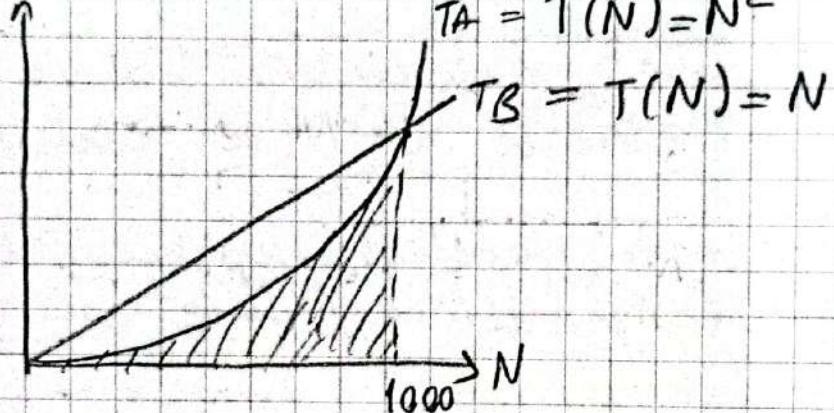
Ornekler

①



$$TA = T(N) = N^2$$

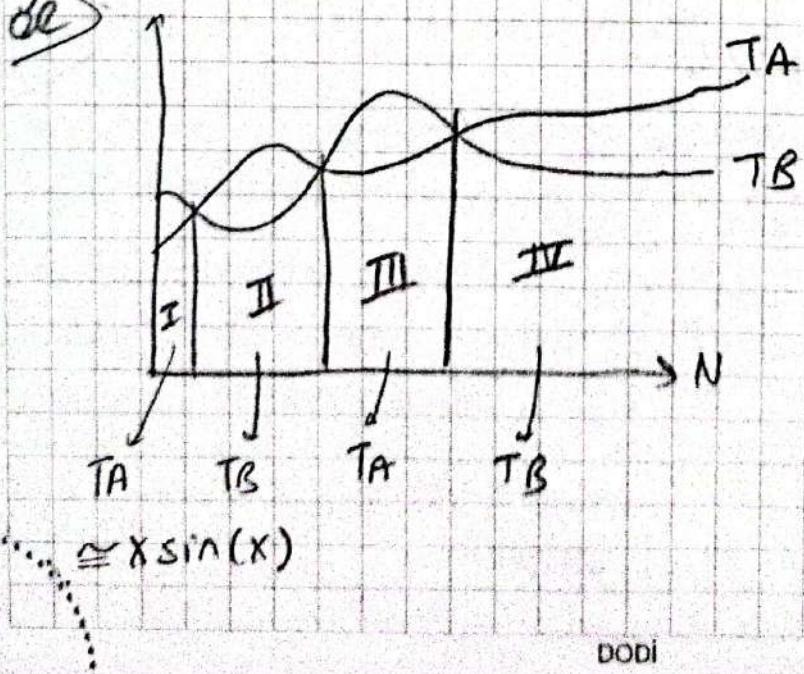
$$TB = T(N) = N$$



$N \leq 1000$ için TA

$N > 1000$ için TB

de)



DODİ

Karsılıklı Genel Fonksiyonlar

Sabit	$O(1)$	
log log	$O(\log \log N)$	Tahminsel Arama, İyi durumlar mis arama algoritmaları - Quick Sort
Logaritmik	$O(\log N)$	
Dogrusal	$O(N)$	Tek Döngü
$N \log N$	$O(N \log N)$	Çoğu sıralama algoritmaları
İresel	$O(N^2)$	İç içe döngü $N < 1000$ için
küpik	$O(N^3)$	$N < 1000$ için
Üssel	$O(2^N)$	Veri boyutu çok küçük olduğunda,

→ class soyuttur, nesne somuttur.

- ⇒  referansı gösterir.

Listeler

- Daha çok bir şekilde veri işlemek gerekligiinde listeler kullanılır.
- Dizi kullanımındaki element sayısının basta verilmesi gerekligi silcisitisi listeler yordimys görür.
- Birde fazla veriyi işlemek için bağılı listeler kullanılır.
- Bağılı listeyle yapılacak 3 temel işlemi ver:

Temel İşlemler
1) Oluşturma (Traversing)

2) Silme (Silme)

3) Arama (Arama)

Yordim
4) Silme İşlemi
5) Düzenleme İşlemi

Bir ya da daha fazla
element ile işlem yapma

Basten silme
Ortadan
Sondan "

Sıralama
Düzenleme İşlemi

Başa eklem
Ortaya
Sona "

Oluşturma:

Referans: class'la aynı isme sahip yapıya denir.

- ⇒ → new ifadesi yoksutur.

class Hayvan {

 public int ayaksayısı; // Global değişkenler

 public Hayvan () {

 Soyut ("hayvan oluşturucusu"); } }

Hayvan kedi; } // Referans

methods
constructors
ve diğer bri

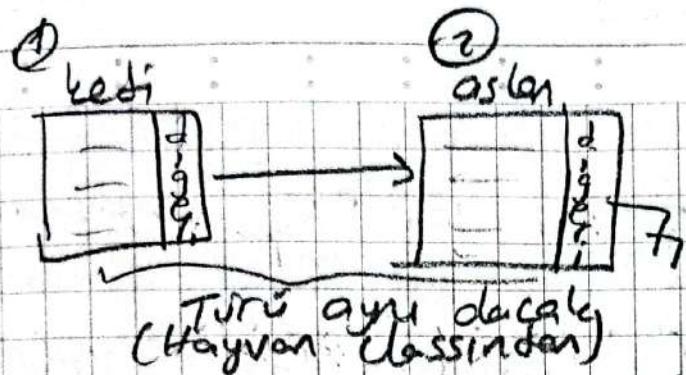
① (Hayvan kedi = new Hayvan());

 kedi.digeri = null; // hiçbir teperi göstermek istemediginde
 null yaparız.

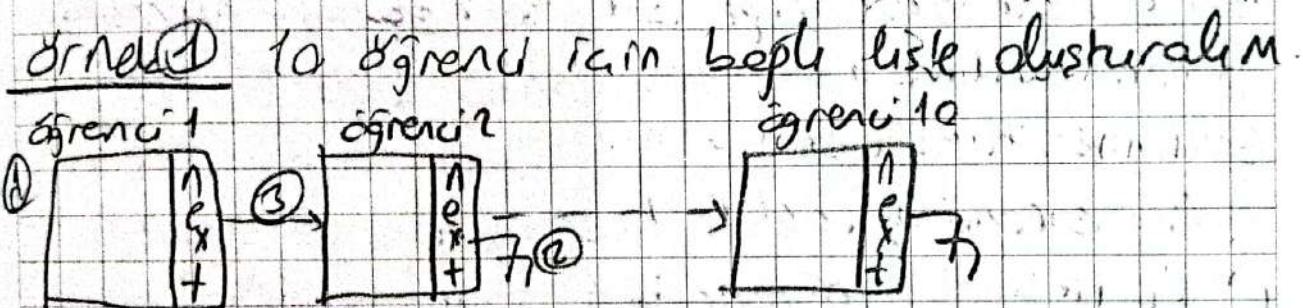
② (Hayvan aslan = new Hayvan());

 aslan.digeri = null; DODI
 kedi.digeri = aslan;

Aynı liste türlerini birbirine cevaplayabiliriz.



NCT: Liste, bir den fazla elemandan oluşur -
tacapi zaman butün elemanların aynı türden
olması esastır.



```
class ogrenci {  
    String ad, soyad, no;  
    int veri;  
    ogrenci next;}
```

psum {

ogrenci yedek;

① ogrenci bir = new ogrenci();

yedek = bir;

② bir.next = null;

for (int i = 1; i <= 9; i++) {

③ bir.next = new ogrenci();

bir.next.next = null;

bir = bir.next; }

bir = yedek; // biri başa alındı tekrar.

}

*NOT: Listeleri oluştururken, özellikle tek yönli bağılı listelerde, baştaki elementin tutulması önemlidir. Çünkü referans ile birlikte baştaki elementin erişilememesi problemi ortaya çıkar. Burun için 2. barten de 3. bir referans ilk elemenin yerleştirilir.

Q2) 5 tane öğrenciye ait ad-soyad - numara ve veri yapıları notu tele yönli bağılı liste şeklinde tutuluyor.

- Bu işlemi gerçekleştiren kodu yazınız.
- Bu listedeki öğrencileri listeleyiniz.

class ogrenci {

 String ad, soyad, no;
 int veri;

 ogrenci next;}

public class ANA {

 psvm{

 ogrenci bir = new ogrenci();

 ogrenci yedek; yedek = bir;

 Scanner k = new Scanner(System.in);

 ① bir.next = null;

 ② (bir.ad = k.nextLine(); bir.soyad = k.nextLine();
 bir.no = k.nextLine(); bir.veri = k.nextInt());

 for (int i=1; i<5; i++) {

 bir.next = new ogrenci();

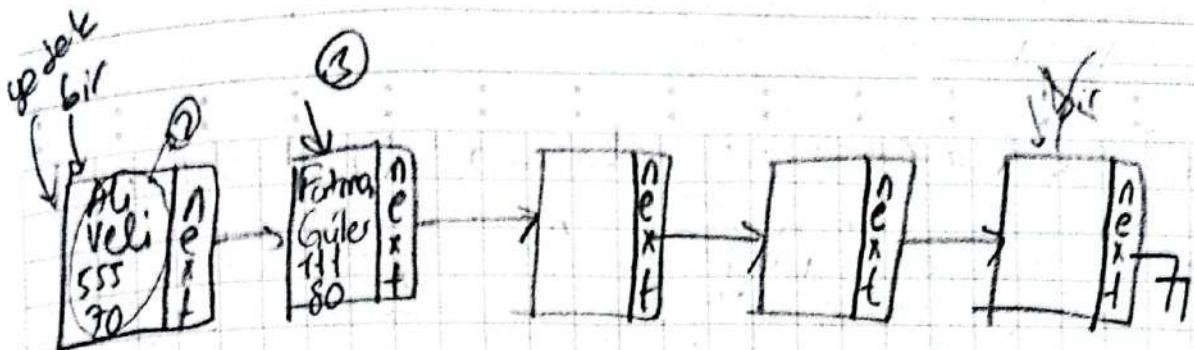
 bir.next.next = null;

 ③ bir = bir.next;

 bir.ad = k.nextLine(); bir.soyad = k.nextLine();

 bir.no = k.nextLine(); bir.veri = k.nextInt();

 } bir = yedek; } // basvurulma



while (bir.next != null) {

Sout(bir.ad + bir.soyad + bir.no + bir.veri);
bir=bir.next;

}

bir=yedek;

b) public static void goruntule (ogrenci bir) {

ogrenci yedek=bir;

while (bir.next != null) {

Sout(bir.ad + bir.soyad
bir=bir.next;

bir=yedek; }

c) A sikkundas uretilen dizinin sonuna bir element ekleyen pr. yapınız.

a) Sona ekeme (Bir elemen) (* Ayni turden elemat gorunduler)

1. asama Son elemene git

while (bir.next != null) {
bir=bir.next;

2. asama Sona bir elemen ilave et

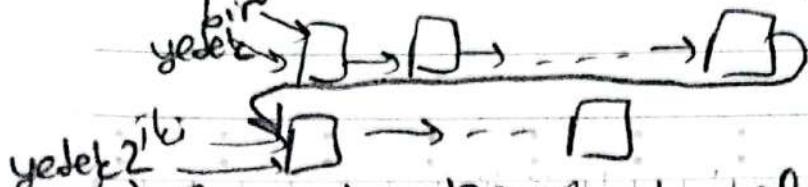
bir.next = new ogrenci();

bir.next.next = null;

bir=bir.next;

3. asama Okuma bir=yedek; // BASA ALMA

new yoluyla kesinlikle referansları.



yedek \rightarrow [] \rightarrow [] \rightarrow []

yedek \rightarrow [] \rightarrow []

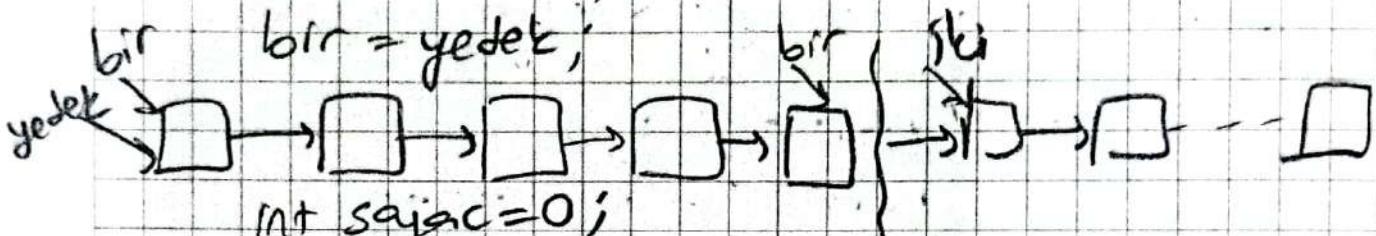
b) Sonra birden fazla eleman (liste) ekleme

```
public void ekle (ogrenci bir, ogrenci *ki){  
    ogrenci yedek = bir;
```

```
    while (bir.next != null) {bir=bir.next;}
```

```
    bir.next = *ki;
```

```
    bir = yedek;
```

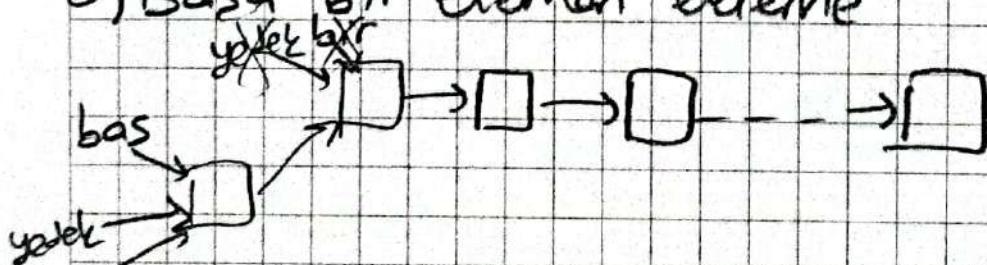


```
int sayac=0;
```

```
while (bir.next!=null) {sayac++; bir=bir.next;}
```

```
bir = yedek;
```

c) Başa bir eleman ekleme



```
bir public void ekle_basa_birEleman (ogrenci bir){
```

```
    ogrenci yedek = bir;
```

```
    ogrenci bas = new ogrenci();
```

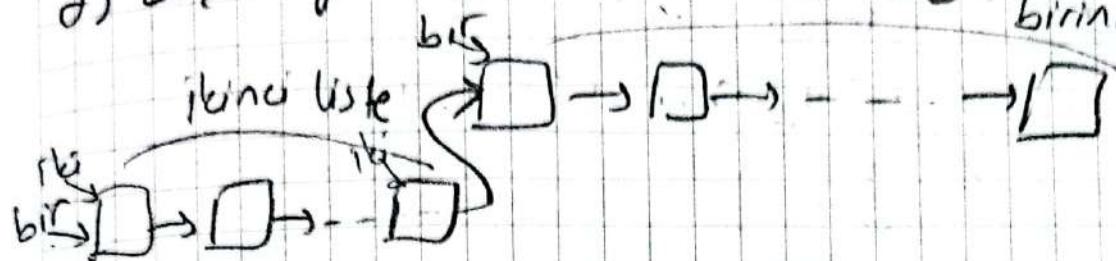
```
    bas.next = bir;
```

```
    yedek = bas;
```

```
    bir = bas;
```

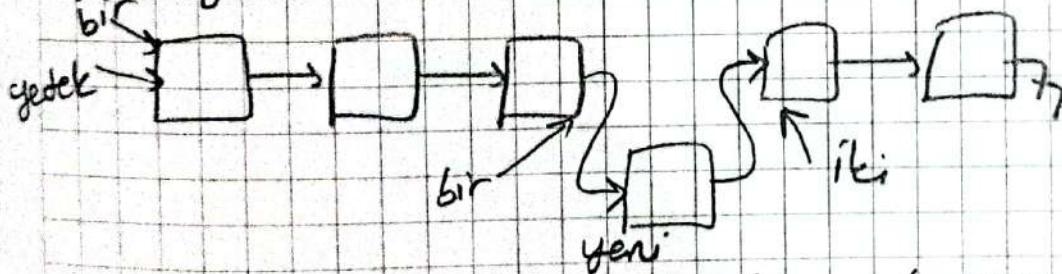
}

d) Basa yeni liste ikile etmeli



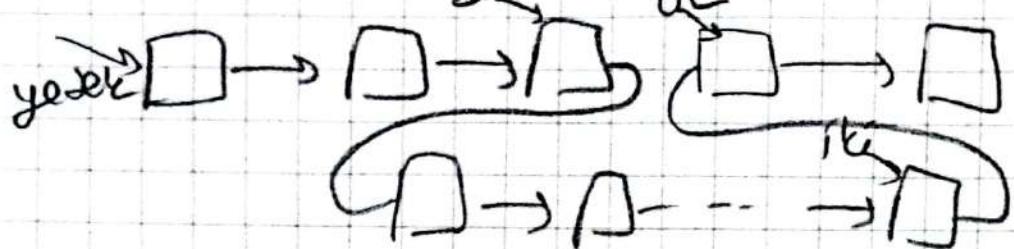
```
public void ekle_basa_liste(ogrenci bir, ogrenci iki){  
    ogrenci template = iki;  
    while (iki.next != null) iki = iki.next;  
    iki.next = bir;  
    bir = template;  
}
```

e) Araya bir eleman eklemeye



```
public void araya_ekle_bireleman(ogrenci bir){  
    ogrenci yedek = bir;  
    for (int i=0; i<1; i++) bir = bir.next;  
    ogrenci iki = bir.next;  
    ogrenci yeni = new ogrenci();  
    bir.next = yeni;  
    yeni.next = iki;  
}
```

f) Araya baska bir liste ekleme

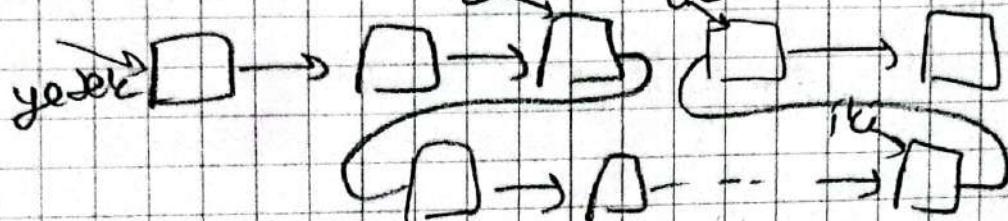


```
public void araya_ekle_liste(ogrenci bir, ogrenci ibi){  
    ogrenci yedek = bir;  
    for(int i=0; i<1; i++) bir=bir.next;  
    ogrenci uc = bir.next;  
    bir.next = ibi;  
    while(ibи.next!=null) ibи=ibи.next;  
    ibи.next = uc;  
}
```

→ Java, hafızada tutulamayan yerler garbage denir.
alımlar atar.

→ C dilinde "free"

f) Araya baska bir liste ekleme

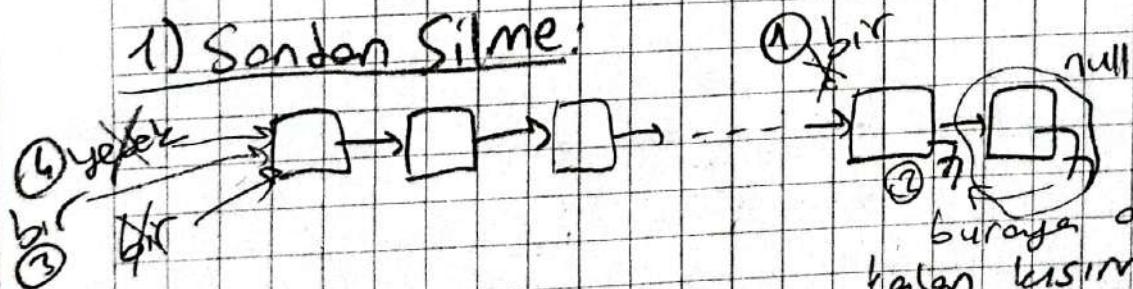


```
public void araya_baska_bir_liste(Ogrenci bir, Ogrenci ibi)  
    ogrenci yedek = bir;  
    for (int i = 0; i < 1; i++) bir = bir.next;  
    ogrenci uc = bir.next;  
    bir.next = ibi;  
    while (ibi.next != null) ibi = ibi.next;  
    ibi.next = uc;
```

}

Tek Yönülu Listedeki Silme İşlemi

1) Sondan Silme:



buraya alacak
takın kısım garbage collection
ile atılır savada.

```
public void sil_sondan(Ogrenci bir){
```

```
    ogrenci yedek = bir;  
    ① while (bir.next.next != null) bir = bir.next;  
    ② bir.next = null;  
    ③ bir = yedek; // Referansı loss almak  
    ④ yedek = null;
```

2) Baslangic Silme:



1. secli bir = bir.next;

2. secli public void baslangic_sil(ogrrenci bir){

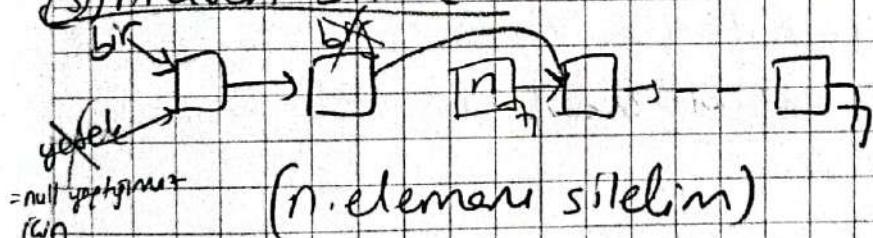
① ogrrenci yedek = bir;

② bir = bir.next;

③ yedek.next = null;

④ yedek=null; }

② Aradan Silme



(n. elementi silelim)

public void sil_aradan(ogrrenci bir, int n){

if (n > elemansayisi(bir)) { Sout ("hata")

System.exit(1); }

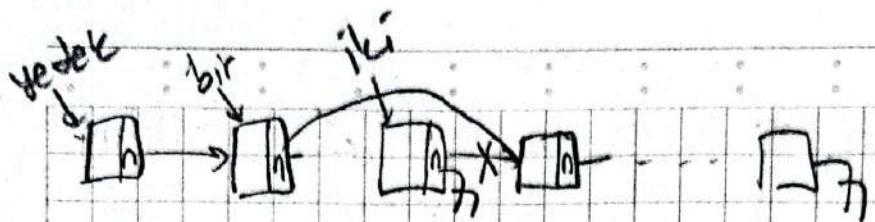
ogrrenci yedek = bir;

for (int i=0; i<n-1; i++) bir=bir.next;
// kendinden bir öncekine gelir.

bir.next = bir.next.next;

bir=yedek; yedek=null;

}



`public void silAradan2 (ogrenci bir, int n) {`

`ogrenci yedek = bir;`

`for (int i=0; i<n-1; i++) bir = bir.next;`

`ogrenci iki = bir.next;`

`bir.next = iki.next;`

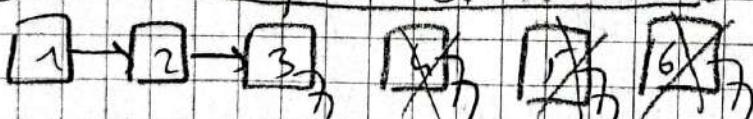
`iki.next = null;`

`bir = null;`

`}`

NOT: Listelerde birden fazla elemen silineceklse eger her bir elementin tek basina birakilmasi gerektir.

4) Sondon birden fazla elemen silme:



`public void sondon_birdenfazla_eleman_silme (int n) {`

`for (int i=0; i<n; i++) sil_sondon(bir);`

`}`

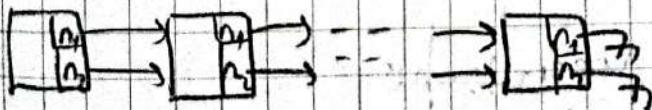
NOT: Birden fazla silme islemi icin, tek bir silme icin yarolen metodlar dengi icinde ugrasmamlidir.

+ Biri algoritmalar static olmadan calismasi.

b) Tek yönlü tek boyutlu liste de eleman sayisi bul

```
public static int elemanSayisi(ogrenci bir){  
    int sayac = 1; ogrenci yedek = bir;  
    while (bir.next != null){  
        sayac++;  
        bir = bir.next;  
    }  
    bir = yedek; yedek = null;  
    return sayac;  
}
```

Tek Yönülu Gift Boyutlu Listeler



```
class Sinif{  
    String ad;  
    Sinif next1;  
    Sinif next2;  
}
```

Gift B. Listelerin kullanildigi yerler:

- Hard disk yönetimi içinde
- disk yerlestirme algoritmasında

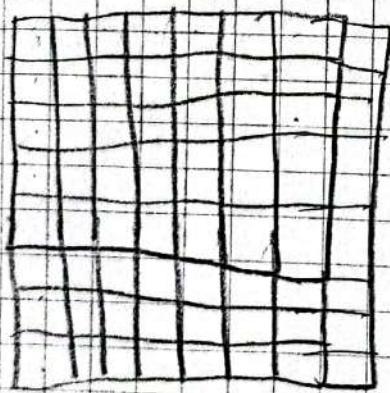
61.5 daha ferha
yer kaplar



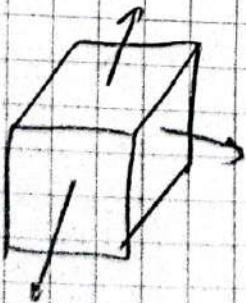
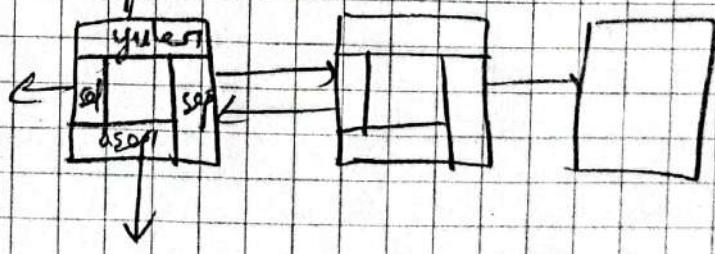
8c) 10 elementli çift boyutlu liste oluşturunuz.

```
public void cift_bapl - liste() {  
    Sınıf ilk = new Sınıf(); Sınıf yedek = ilk;  
    ilk.next1=null; ilk.next2=null;  
    ilk.ad = "Binnur";  
    for (int i=1; i<10; i++) {  
        ilk.next1 = new Sınıf();  
        ilk.next1.ad = ilk.nextLine();  
        ilk.next1.next1=null;  
        ilk.next1.next2=null;  
        ilk=ilk.next2;  
    }  
}
```

Örnek: Satırı satırı, matris yapıları liste ile
oluşturulabilir.



Mesh Yapısı (Grid) Matris



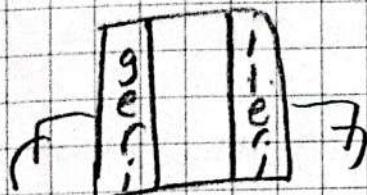
Recursive
Liste Yapısı

3	4	14
7	5	6
8	6	7

Liste

Gift Yönü Listeler

yonları, referansları ters şekilde çizilmiş listendir.

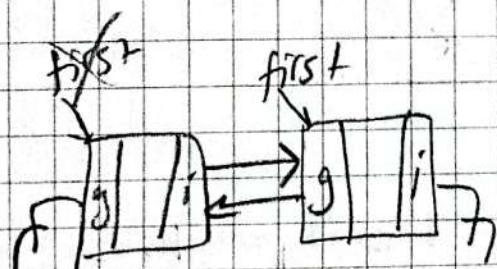


class Bilgi{

==

Bilgi ileri;

Bilgi geri; }



(2) 5 elementli bir gift yönü liste oluşturunuz.

psum{

Bilgi first = new Bilgi();

first. ileri=null; first.geri=null;

Bilgi yedek = first;

for (int i=0; i<5; i++){

first. ileri = new Bilgi();

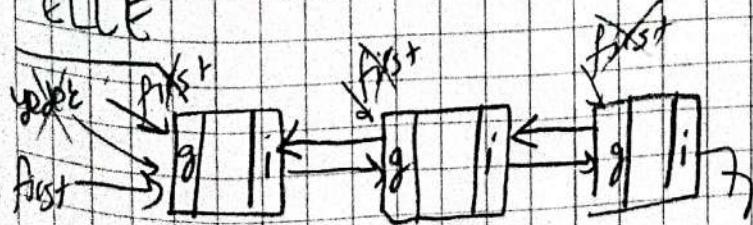
first. ileri. ileri=null;

first. ileri. geri = first;

first = first. ileri;

} first=yedek; yedek=null;

ELLE

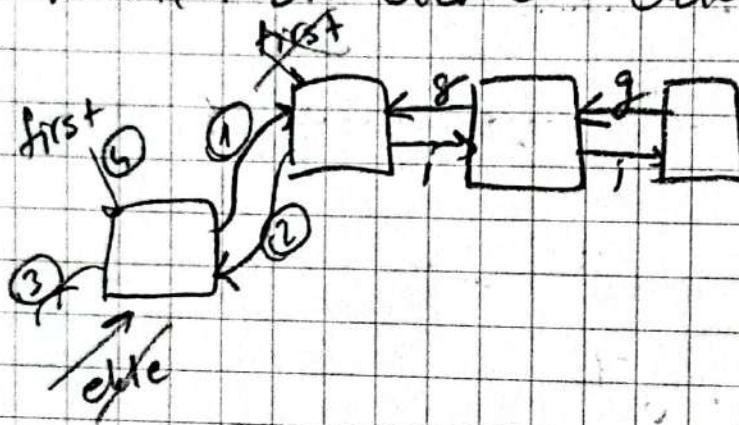


DODI

Gift Yönülu Liste İşlemleri

A) Ekleme

A1) Basa bir eleman ekleme



```
public void ciftyanluliste_basa_ekleme(Bilgi first){  
    Bilgi elde = new Bilgi();
```

① elde.ileri = first;

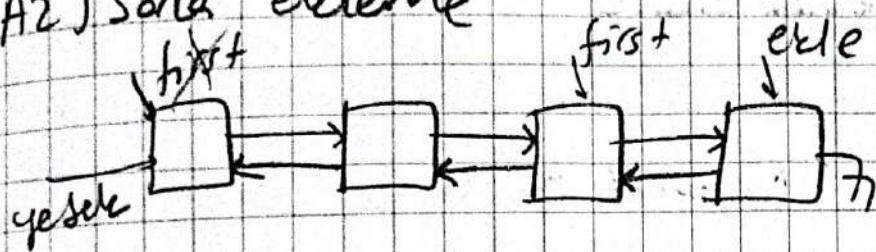
first.geri = elde;

elde.geri = null;

first = elde;

elde = null;

A2) sona ekleme



```
public void sona_ekle_ciftyanlu(Bilgi first){
```

```
Bilgi yedek = first;
```

```
while (first.next != null) first = first.next;
```

```
Bilgi elde = new Bilgi();
```

```
first.ileri = elde;
```

```
elde.geri = first;
```

```
elde.ileri = null;
```

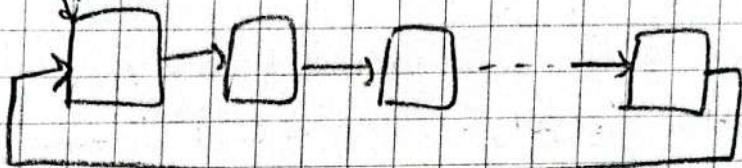
```
first = yedek;
```

```
yedek = null; elde = null;
```

Dairesel Listeler:

Tanım: İşletim Sisteminde, process yönetimi münasebetinde kullanılır.

first Dairesel Tek Yönülu Liste



- * Sondakı elementin referansı baştakini gösterir.
- * Dairesel liste uygulamalarında zaman esastır.

Ölçütürme

8e) 10 elementli içinde tam sayı, string olan dairesel listeyi ölçütürünüz.

class Dairesel {

int t;

String s;

Dairesel next;

psvm{

Dairesel first = new Dairesel();

Scanner k = new Scanner(System.in);

first.t = k.nextInt();

first.s = k.next();

first.next = null; Dairesel yedek = first;

for (int i = 1; i <= 9; i++) {

first.next = new Dairesel();

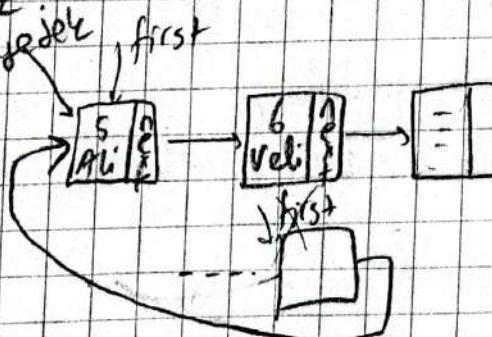
first = first.next;

first.t = k.nextInt();

first.s = k.next();

} (first.next = yedek); // Dairesel liste

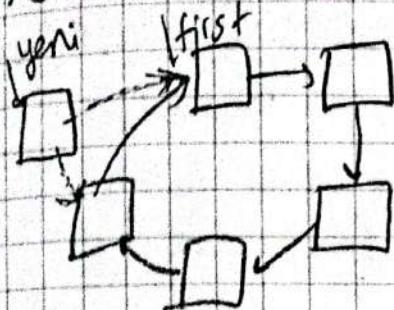
first = yedek;



İlave

Ekleme

→ Basına ekleme → Birkaç şekilde yapılabilir.



Sonuna ekleme

1. Sekli: Listedeki element sayısını bulma

2. Sekli: Yedek bir referans tanımlanır. $first.next = yedek$; gösterdiğinde ilave ederiz.

```
public void ekle_dairesel_bas (Dairesel first) {
```

Dairesel yedek = first;

while (first.next != yedek) { // Dairesel listenin
first = first.next; sonuna gider.

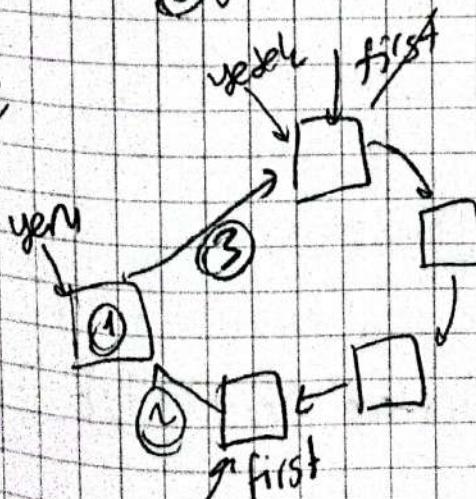
}

① Dairesel yeni = new Dairesel();

② first.next = yeni;

③ yeni.next = yedek;

}



Tek yönlü

82) Dairesel listenin element sayisini bulan
kodu yazınız.

```
public int dairesel_liste_elemansayisi(Dairesel first){  
    dairesel yedek = first;  
    int sayac = 0;  
    while(yedek.next != first){  
        sayac++;  
        yedek = yedek.next;  
    }  
    yedek = null;  
    return sayac;  
}
```

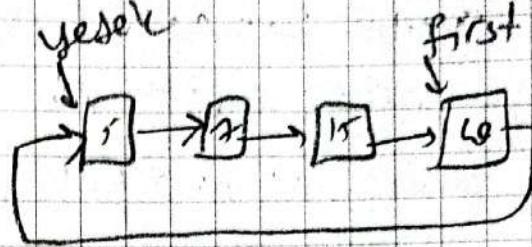
83) Klavyeden rastgele sayılar giriliyor. Bu sayılar
dairesel liste içinde tutuyoruz. Giren sayı
999 ise dairesel listeyi oluştururan kodu yazınız.

```
class sayi{  
    int t;  
    sayi next; // Referans  
}  
public class ANA{  
    public static void main(){  
        Scanner k = new Scanner(System.in);  
        int a = k.nextInt();  
        if(a == 999){  
            System.out.println("dur"); System.exit(1);  
        }  
        sayi first = new sayi();  
        first.t = a;  
        sayi yedek = first;
```

```

while(true) { // for(; ;) // while(true){
    a = k.nextInt();
    if (a == 999) break;
    first.next = new Sayi();
    first = first.next;
    first.t = a;
}
first.next = yedek;
first = null; // first kaldi
}

```

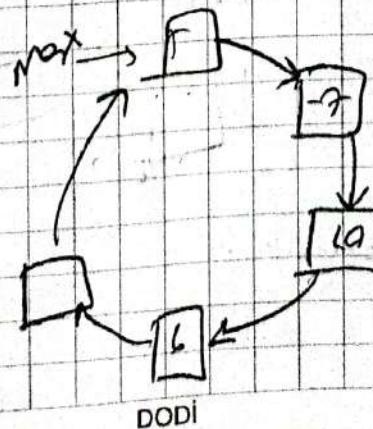


sayı Oncelikle ennekteli dairesel liste deki sayıyı bulan metodu yepinir. enb kaçinci node?

```

public void max_bul(Sayi first){
    Sayi yedek = first; int kacinci = 0, n = kacinci;
    int max = first.t;
    first = first.next;
    while (first.next != yedek) {
        if (max < first.t) {max = first.t; n = kacinci;}
        first = first.next;
    }
    System.out.println(max);
    System.out.println(n);
}

```



Bir dairesel listenin basina ait metodu yazınız.
// içerdeki referansları değiştirecek
// içerdeki context'ı değiştirecek

```
public int max_bul_dairesel (sayi first){
```

```
    =  
    =  
    return max; }
```

```
public void max_basa_ol (sayi first){
```

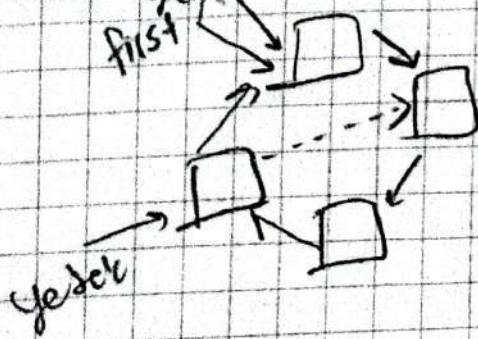
```
    int a = max_bul_dairesel (first); // Max değeri  
    sayi yedek = first; // a'dadir.
```

```
    while (first.t != max) first = first.next; // max  
    elemente  
    gitmek  
    için  
    int temp = yedek.t;  
    yedek.t = first.t;  
    first.t = temp;
```

}

c) Dairesel liste silme işlemi

Baştanı silme = herhangi bir yedekini silme



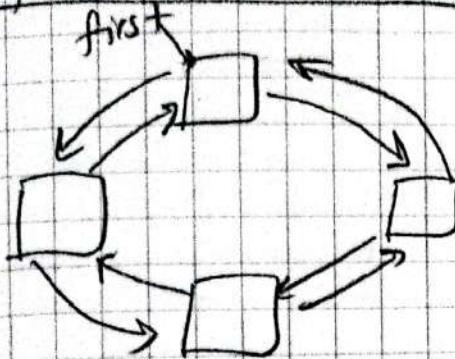
```
sayi yedek = first;  
while (yedek.next != first) {  
    yedek = yedek.next;
```

```
}  
yedek.next = first.next;  
first.next = null;
```

Baştanı
silip kopartık,
büzgümüz de bitti
gilmediğiz.

DODI

Cift Yonlu Dairesel Listeler

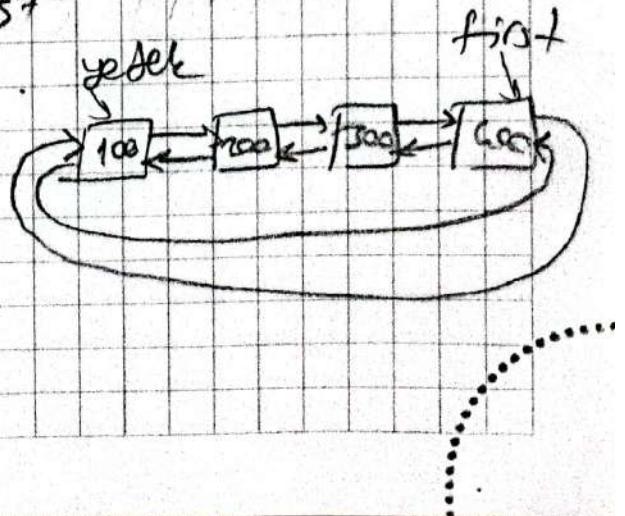


```
class cift_yonlu {  
    int t;  
    cift_yonlu next;  
    cift_yonlu back;}
```

Olusturulmasi

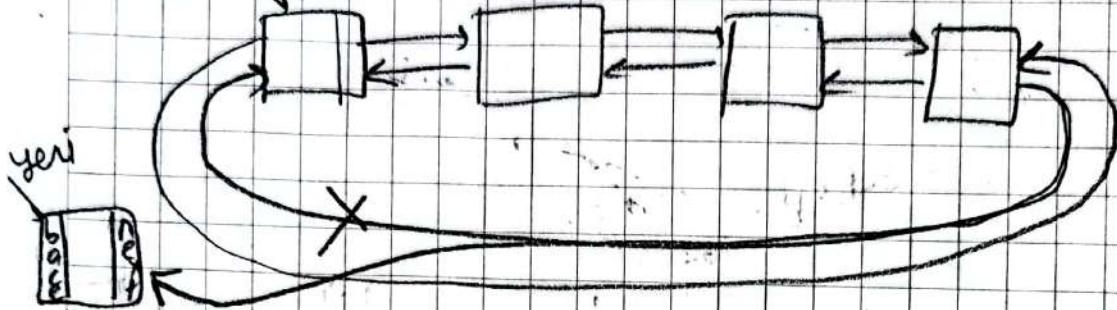
6 elementli cift yonlu dairesel bagli liste
psvm

```
cift_yonlu first = new cift_yonlu();  
first.t=100; first.next=null; first.back=null;  
cift_yonlu yedek = first;  
for (int i=1 ; i<=3 ; i++){  
    first.next = new cift_yonlu();  
    first.t = 100*(i+1);  
    first.next.back = first;  
    first = first.next;  
}  
first.next = yedek;  
yedek.back = first;
```

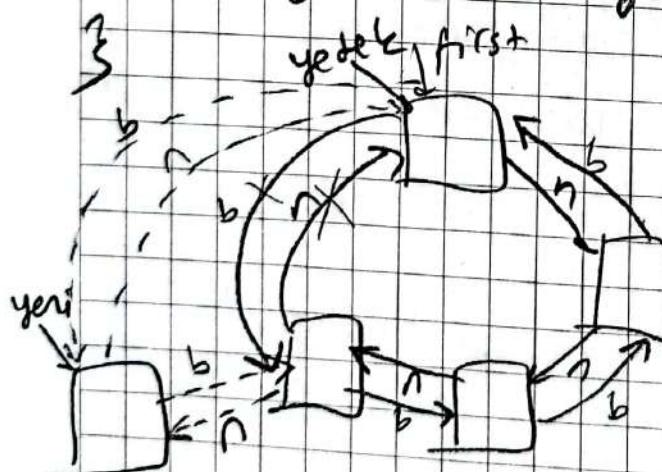


Cift yonlu olmasi hareket esnekligi saglar.

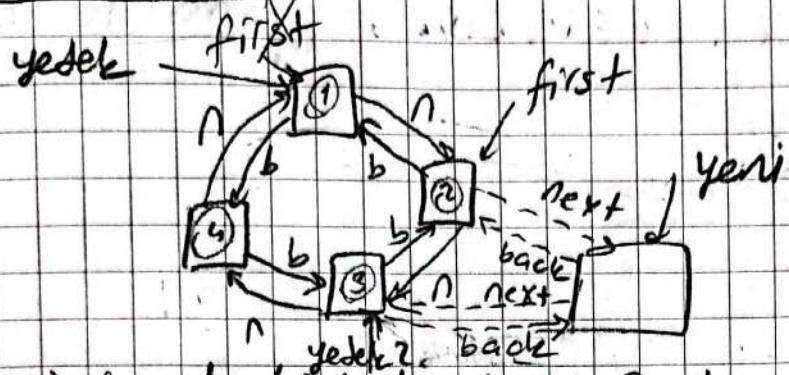
A) Ekleme (Bas kisminda = Son kisminda)



```
public void ekle_basa_cift_yonlu (cift_yonlu first){  
    cift_yonlu yeni = new cift_yonlu ();  
    first.back = yeni;  
    cift_yonlu yedek = first;  
    while (yedek.next != first) yedek = yedek.next;  
    yedek.next = yeni;  
    yeni.next = first;  
    yeni.back = yedek;  
}
```



b) Araya eklemeye:



82) Bir dairesel liste de 2. ve 3. elemanı arası
için eklenecek yeni yerini yazınız.

```
public void araya_ekleme_cift_yanlı (cift_yanlı first, )  
    int n
```

```
cift_yanlı yedek = first;
```

```
-> for (int i=0; i<n; i++) first = first.next;
```

```
cift_yanlı yeni = new cift_yanlı();
```

```
cift_yanlı yedek2 = first.next;
```

```
first.next = yeni;
```

```
yeni.back = first;
```

```
yeni.next = yedek2;
```

```
yedek2.back = yeni;
```

```
yedek2 = null;
```

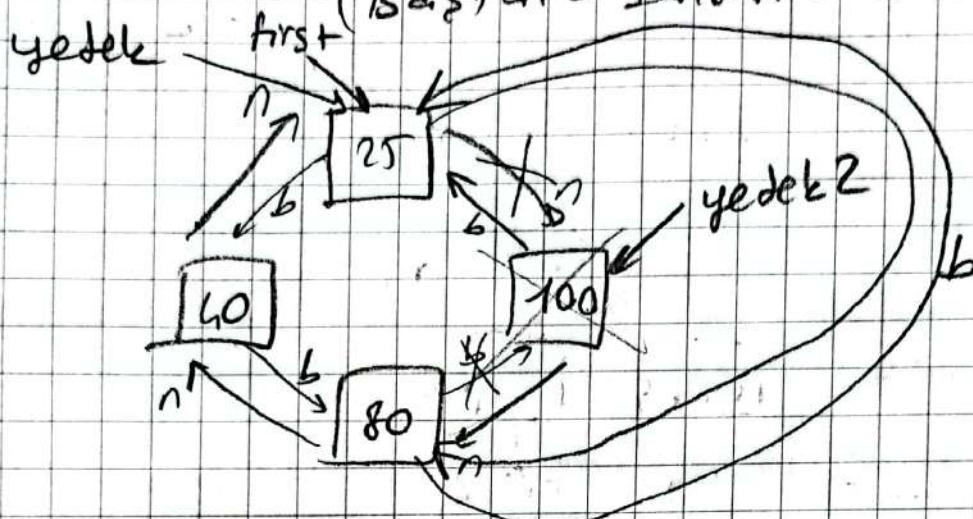
```
first = yedek;
```

```
yedek = null;
```

```
}
```

Silme işlemi : (GENELDIR)

(Bastır = silin = aranın)



82) Bir dairesel liste de öğrencilerlere ait Veri-yapıları notu tutuluyor. Öğrenci notu 100 olan kişiyi, bu dairesel listeden silmek için program yapınız.

class cift_yanlı {

 int not;

 cift_yanlı next;

 cift_yanlı back;

} public void clear_cift_yanlı(cift_yanlı first){

 cift_yanlı yedek = first;

 while (first.next != yedek){

 if (first.not == 100){ cift_yanlı yedek2 = first; }

 first = first.next;

 }

 first.next = yedek2.next;

 yedek2.next.back = first;

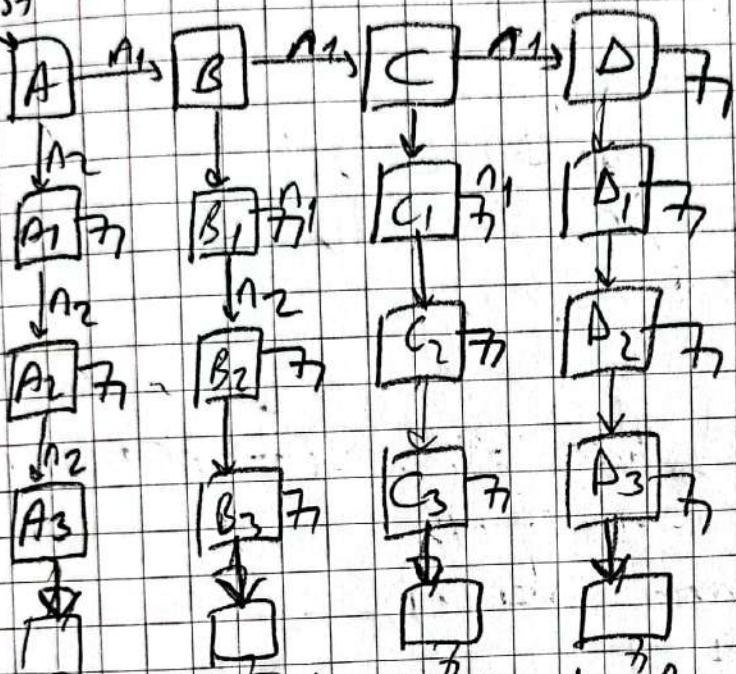
 yedek2.next = null; } Tersitlik önacılı

 yedek2 = null;

8zel Yapılar

Kural: Listeler oluşturulukesi aynı class tipi kullanılır. & Aynı tip olmasa bağlantı kuramazsınız.

① First



Uygulama Alanları: "Hash" fonksiyonlarının
formülündeki yapılarında bu tip listeler kullanılır.

Hash
mod(3)

$$21 \text{ mod } 3 = 0$$

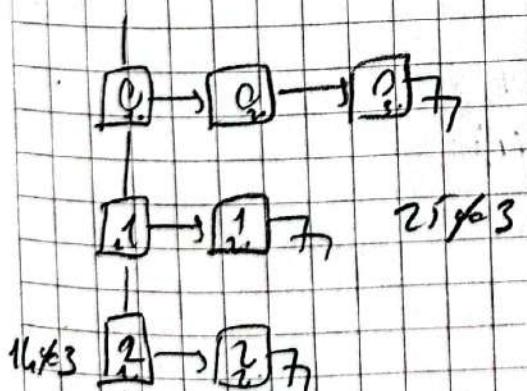
$$24 \text{ mod } 3 = 0$$

grafik sistemlerinde
oyunlarda

NOT: 2D veri yapları

veri tutmada, grafik sistemlerinde, oyunlarda, şifreleme algoritmalarında sıkılıkla kullanılır.

Özellikle veri yapılarında hash fonksiyonlarında sıkılıkla kullanılır.



- 2D veri yapıları -

class OZEL1{

String s;

OZEL1 n1;

OZEL2 n2;

}

psum { int n=4, m=5; }
ssum satf.

OZEL1 first = new OZEL1();

first.n1=null; first.n2=null;

olustur_ozel(first, n, m);

}

public static void olustur_ozel (OZEL1 first, int n, int m){

OZEL1 yedek = first;

for (int i=0; i<n; i++) {

first.n1 = new OZEL1();

first = first.n1;

first.n2=null; }

first.n1=null;

first=yedek;

OZEL1 yedek2 = yedek;

while (first.n1!=null) {

for (int i=0; i<m; i++) {

first.n2 = new OZEL1();

first = first.n2;

first.n1=null

first.n2=null;

first=yedek;

yedek=yedek.n1;

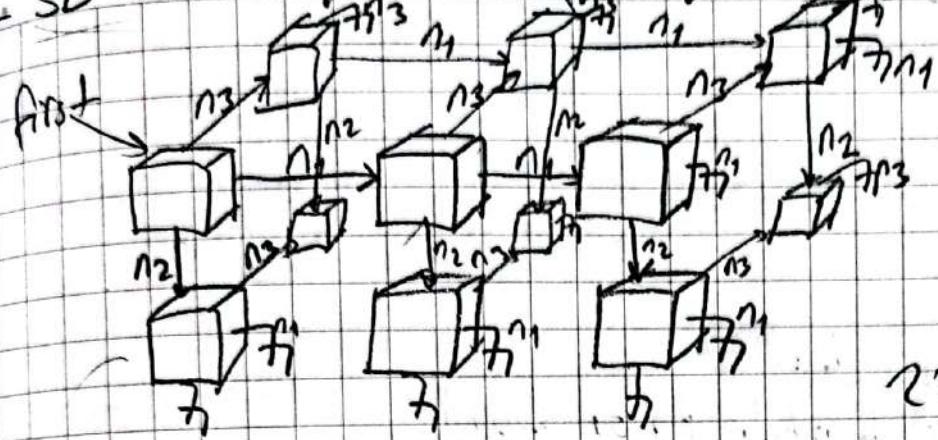
first=first.n1;

...
...
...
33

DODI

(özel yapılar sınırlıda çok ama önemli)

3D Veri Yapısı - (tek yönlü)



$2 \times 2 \times 3$

Kullanılmış yerler

Network işlemleri

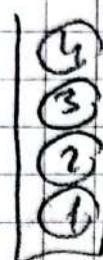
Çift yönlü alanları → dajitik programlamada,

↪ 1 bilgisayarda 12 birimlik
İş yapacapınıza,
12 bilgisayarda 1 birim iş
yaparız.

↪ her bir node'u bilgisayar gibi düşünübüriz.
grafik olarak düşünügümüzde her birini
renklendirince node olarak düşünübüriz.

+ satırı

YIGIT (STACK)



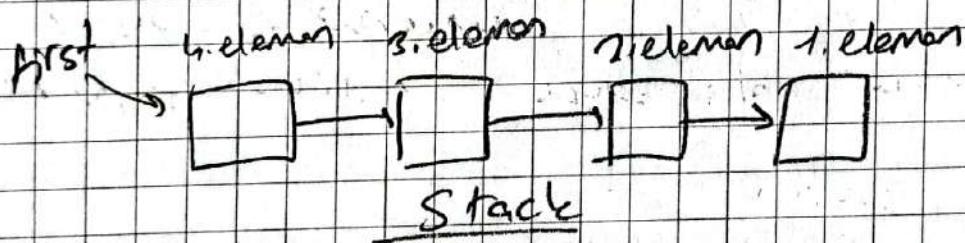
LIFO

(Last In First Out)

(1. elemenin silinmesi için diğerlerini atarız)
Listedeki gibi deşti

NOT 1: Yigit ve kuyruk tek yönlü örel listedelerdir.

NOT 2: DIGITAN elemen silme işlemek temel, tek
yönülü listenin başına elemen eklemektir.



NOT 3: Tek yönülü listede baştaki elementin işlem yapmak (silme, okuma) stack teknik ile elemeni silmemek temeldir.

Kullanıldığı yerler:

1) Recursive metodlar çalışırken tampon - bellek (buffer) şeklinde stack kullanılır.

2) Bazi algoritmalar örneğin polish notation, ağacalar üzerindeki bazi işlemler yine stack manzıuya gösterilir.

→ Stack'in hem yazılım hem deneyim mesleği vardır.

Stack İşlemleri

1 → Ekleme (

2 → Çıkarma (Elemanı ekle etme)

3 → Silme (Sondan silme)

→ Stack'te araya eleman eklenme / çıkarma yok
sadece bas ve sondan.

Kuyruk (Queue)

FIFO çalışır.

(ilk giren, ilk çıkar)

Kullanıldığı yerler:

→ Bankada para çekmekten sonra bekleyen kisilerin
yapısı gibidir.

→ İşletim Sisteminde çok sık kullanılır. Tek ve çok
seviyeli obranızın bilgisayarda kuyruk yapısı
varır.

Tek seviyeli → yarıcı işlemleri

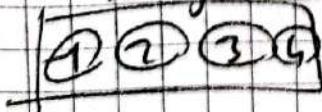
çok seviyeli → yapacağınız işleri önceki sırasına
göre alır.

→ İşletim Sisteminde processleri çalıştırırken yarıcı
hafızada aktif olan kodlar kuyruk şeklinde
seviyelere göre çalıştırılır.

→ Bilgisayar portlarındaki kümük hafızalar kuyruk
mentiğiyle çalışır.

→ Bilgisayar hardware kısmında kuyruk küteler
genellikle (?)

kuyruk



DODI

A) Eleman Oluşturma = Standart

* Tek yönlü listedir.

a) Eleman silme = Listenin ilk elementini silme *

b) Eleman ekleme = Listenin sonuna eleman eklemeye
isleridir. *

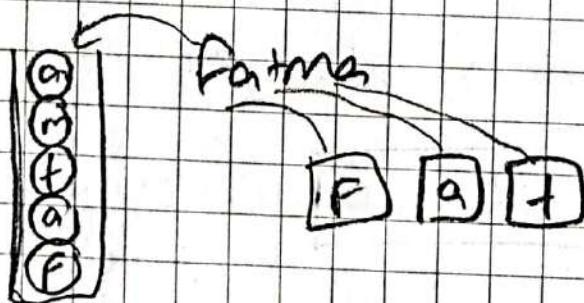
c) Kuyruğun tamamını silme = Listeyi tamamen
bosaltmak / silmektir.

Kuyruktaki elemanları silmek boston itibaren
bütün elemanları silmektir.

D) Adının karakterlerini tek yönlü listeyle

a) kuyruk şeklinde

b) Yığın şeklinde tallalı ezen grafiğe uygunlasmam
animasyon şeklinde yapınız.



(Applet olarak yapabilirsin)

En iyi durum \rightarrow Magn

Hash Yapisi (Hashing)

Tanım: Bilgi sayısında bütün veriler sayısal olarak tutulur. Verinin tipi ne olursa olsun gerçek ondalıksız, gerçek pozitif gerçek negatif gereklisi karakterlesiz daima üzerinde bütün hepsi sayısal tutulur.

Bu verileri tutmanın en kötü durumu bütün veriler aynı yerde ayırmaktır. Ancak pratikte bu mümkün degildir (milyonlarca veri olabilir).

Bunu çözmek için hashing (karma) kullanılır

İşlemler

Ekleme

Silme

Güncelleme
Arama

Yöntem

1 \rightarrow Linear Arama

$O(n) \rightarrow$ en kötü durum

2 \rightarrow Skili Arama \rightarrow (β_0 ve γ_0)

(Sıralı olacak) $\rightarrow (K \rightarrow \beta)$

$\log_2 N$

3 \rightarrow Dogrulanın arama - (en hızlı)

$O(1) \rightarrow$ diziler $\rightarrow [100]$

\hookrightarrow hizlazza doğrudan erişim söz konusu

1) Hash fonksiyonu seçilir.

$\boxed{1 \% 10}$

Hash table

0	100	\rightarrow red	$\text{or} \rightarrow$
1	61		
2	72	\rightarrow 102	\rightarrow 22
3	83	\rightarrow 133	
4			
5			
6	86	\rightarrow 96	
7			
8			
9			

$$61 \% 10 = 1$$

$$72 \% 10 = 2$$

$$86 \% 10 = 6$$

$$96 \% 10 = 6$$

$$102 \% 10 = 2 \quad 83 \% 10 = 3 \quad 133 \% 10 = 3$$

$$22 \% 10 = 2$$

$$100 \% 10 = 0 \quad 200 \% 10 = 0$$

Linked List

(Dizi) C(1)

Ana tabloyu boş bırakmayacaksız
sözleme doldurmayı.

Quadratic \rightarrow 7. tereceten
kübik \rightarrow 3. "

→ gitdecekti tablo doluyse liste oluşturur.

1. Yöntem Separate addressing (Ayrık Adresleme)

Ayrık adreslemenede tabloyu tam dolduramadık.

① 22'ye erişmek istedigimizde önce 2'ye geldik
sonra linear search $O(1) + O(n)$ -
 $O(n)$

> Oysaki $O(1)$ de çalışmaşını istiyorduk

⊕ Ayrık adreslemenin avantajı hızla çalışmaşır

⊖ aynı hash fonksiyonu kullanırsa sahip değerler
için liste oluşturulması gereklidir.

2. Linear Probing Yöntemi

$$(h(x) + i) \% S \quad h(x) = \%(\text{mod } 10)$$

loop

0	78
1	
2	22
3	32
4	
5	45
6	65
7	105
8	68
9	78

$$22 \% 10 = 2 \quad i=0 \text{ için}$$

$$32 \% 10 = 2 \quad i=1 \text{ için}$$
$$(i+1)=3. \text{ yerde}$$

$$68 \% 10 = 8 \quad i=0$$

$$78 \% 10 = 8 \quad i=1 \rightarrow i+1=9$$

$$18 \% 10 = 8 + (i=2) = 10 \% 10 = 0$$

$$45 \% 10 = 5$$

$$65 \% 10 + 1 = 5 + 1 = 6$$

$$105 \% 10 = 5 + 2 = 7$$

⊗ sayıların gelis sırası önemli

⊗ tablo dolduktan sonra daha
fazla yerlestiremiyoruz.

⊕ tablonun tamamen doldurulabilir
tablonun tam doldu olması
yeterlidir.

$$\Rightarrow h(x) = [x^2 + 9x5] \quad (x^3 + x^2 + x) \neq x$$

Quadratic Probing Yontemi

$$h(x) + i^2 \mod 5$$

$$h(x) = 5x10$$

10 tane deger

olar.

& alacaklar
teğeler sırılıdır.

0	73
1	42
2	12
3	32
4	85
5	45
6	65
7	89
8	92
9	75

A) $65 \mod 10 = 5$

$$65 \mod 10 = 5 + 1 = 6$$

$$75 \mod 10 = 5 + 2^2 = 9$$

$$85 \mod 10 = 5 + 3^2 = 16 \mod 10 = 6$$

B) $12 \mod 10 = 2 + i^0 = 2$

$$32 \mod 10 = 2 + 1^2 = 3$$

$$62 \mod 10 = 2 + 2^2 = 6$$

$$2 + 3^2 = 11 \mod 10 = 1$$

NOT : Tablo doldurulan sonra baska deger yerlestirilmelidir.

* hash tablo oras $O(1)$ de calismalıdır.
en hizli olur.

1) Liste maximum bulma

DNS Bir sınıfındaki veri yapılarının olan öğrencilerin notları listede (tele yönü baglı) tutuluyor
Bu listeyi notla göre sıralayınız ($K \rightarrow B$)

```

class ogrenci {
    int not;
    ogrenci next;
}

public class ANA{
    psvm(){
        Scanner k = new Scanner(System.in);
        ogrenci first = new ogrenci();
        first.not = k.nextInt();
        first.next = null;
        for (int i=1; i<5; i++) {
            ogrenci yeni = new ogrenci();
            yeni.not = k.nextInt();
            yeni.next = null;
            first.next = yeni;
            first = yeni;
        }
        first = yedek;
        while(first!=null) {
            if(first.next.not > first.not) {
                int temp = first.next.not;
                first.next.not = first.not;
                first.not = temp;
            }
            first = first.next;
        }
        first = yedek;
    }
}

```

FYS
AL
50 77

class sinif {

String ad_soyad;

int veri_not;

int eleman_sayisi=0; sinif next;

public sinif (String isim, int not) {

this.ad_soyad = isim;

this.veri_not = not;

this.next = null;

}

}

public ANA{ sinif yeni=null; //kompozisyon

psvm(){

olustur(5);

maximum_bulma(yeni);

}

public static void olustur(int n){

Scanner k = new Scanner(System.in);

String s; int not;

s=k.nextLine(); not=k.nextInt();

sinif first = new sinif(s,not);

for(int i=0; i<(n-1); i++){

s=k.nextLine(); not=k.nextInt();

first.next = new sinif(s,not);

first=first.next;

first=yedek; yedek=null;

yeni=first;

}

postfix-infix tabii değil
islemelerin ina
altersa sırasıda secileni olur

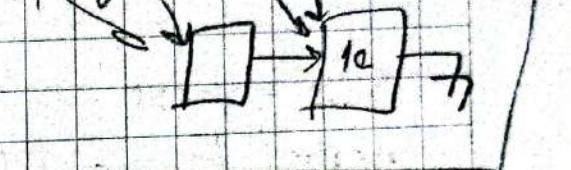
```
• public static void maximum_bulma(Sinif first){  
    int max = first.veri-not;  
    String s = first.ad-soyad;  
    Sinif yedek = first;  
    first = first.next;  
    while(first!=null){  
        if(max<first.veri-not){  
            max = first.veri-not;  
            s = first.ad-soyad;  
        }  
        first = first.next;  
    }  
    cout(max + " " + s);  
}
```

20 elementli yigit (stack) yapisi oluşturmak
yapıtları elementları silerek ip yigitlerin sırasını
kodu yapınız.

```
class yigit{  
    int data;  
    yigit next;
```

```
}  
public class ANAS  
psvm(){
```

if(yigi.top == first)



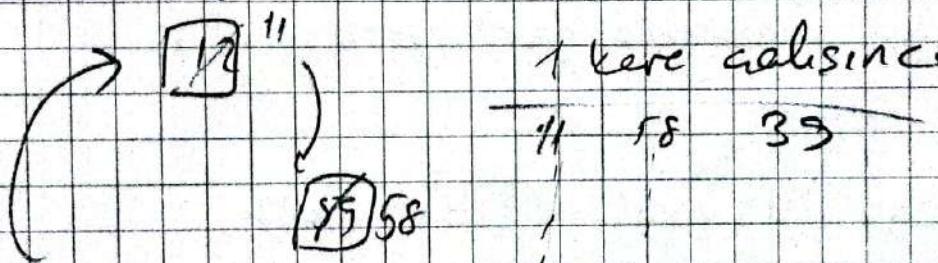
ÖDEV 5

① 10 işlemci bir bitgisayardan işler calistirilmak
isteyen Calisma Seteli dairesel liste ti. Her
bir işleminin calisacagi sure 10 ile 100 arasinda
ms cinsinden dir. rontom atayacak bu

a) bu dairesel Uzleyi olusturen PL.

b) dairesel listedeki butun işlerini bitmesi için
yani listenin silinmesi için gerekli kodu silin. (threat vulnerable bu işlerin geciklestirilmesi.)

Not: her bir işlemin min calisma süresi
1 ms dir. her bir node' 1 ms calisacak
silmesi için gerekken sure



[Kod] o o
59 58
bitenleri listeden siler.
Toplam sureyi: yar.

② Bigrayların yanici sin toplam 5 adet dosya
gonderilecektir. Dosyalarin boyutları 100, 200, 300
50, 20 birimidir. Her bir birim yanici
1 ms'de yesdirilir.

a) butun dosyalarin yesdirilmasi için gerekli
den kodu yaziniz.
(kuyruk olusturduktan sonra siliceg?)

b) her bir dosyanin yesdirilmasi için
gerekli den silreyi ekranan yaziniz.

S b

- ③ 4,0 elemenli dairesel çift yönlü liste üzerinde;
- bulistedeki en uzun karaktere sahip olan element
 - en kısa karaktere sahip olan element bulunur.
 - en uzun karakter ile en ~~kısa~~ karakteri yer değiştirilen kodu yazınız (icerik değiştirme)

NOT: Hash'te karakter gelmesse ascii kodları toplanıp tabloya yerler.

Hash tablosunda karakter ve string'ler ascii değerleri toplamına göre yerleştirilir.

c) Ali

$$97 - 1112 + 105 = 282 \rightarrow \text{Hash}$$

sonra mod alınır.

* mod fonksiyonunu büyük sezmek önemli ama çok da büyük olursa işlevi kalmaz.
(mod 1.000.000.000!!)

AGACLAR

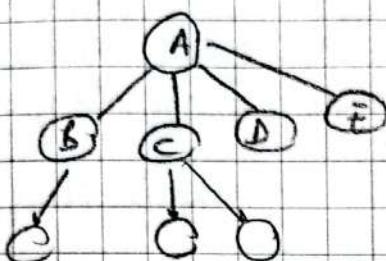
* Ağaclar temel veri modellerinden biridir.

Kullanım yeri: OS, dosya organizasyonu, ve veri tabanlarıdır.

* Ağaclar tek yönlü grafların özel bir halidir.

→ veriye erişimle ilgili, kütüye erişimle ilgili.

* Veriye tutulması

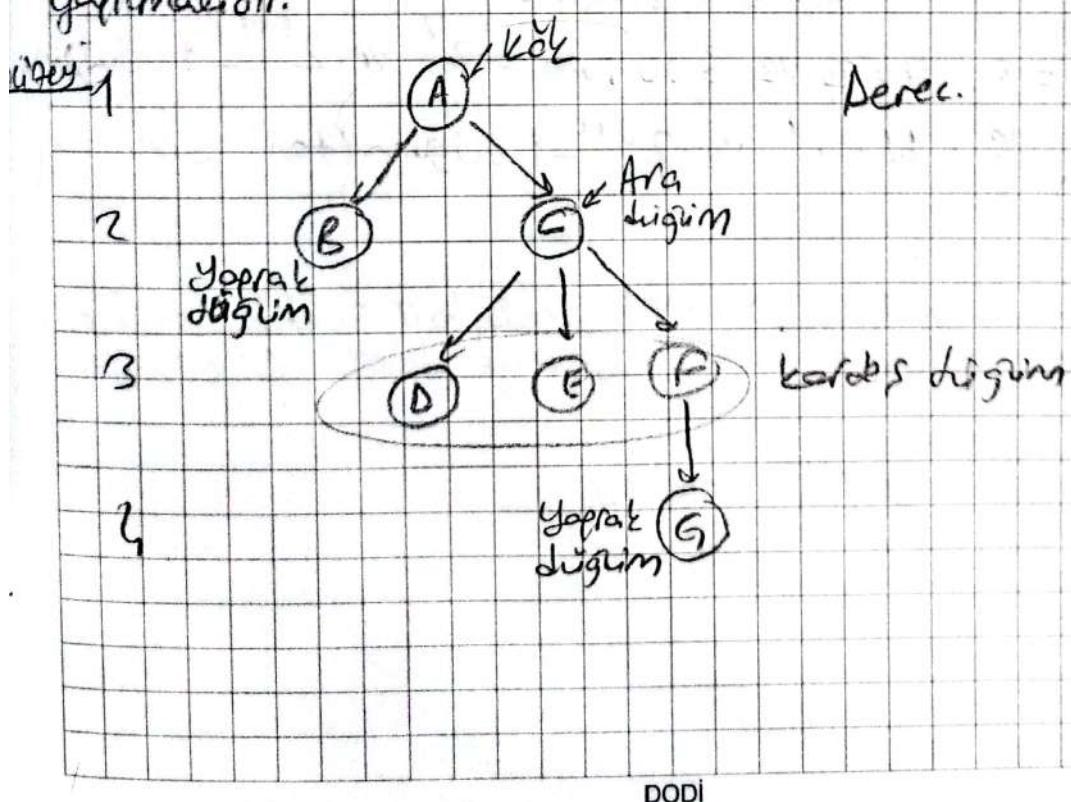


Tek yönlü graflar → Tree

* Verilerin güvenli bir şekilde tutulması, erişilmesi işlemlerinde ağac yapıları kullanılır.

(B. Slayt) NOT Disk yüzeyinde verilerin tutulması

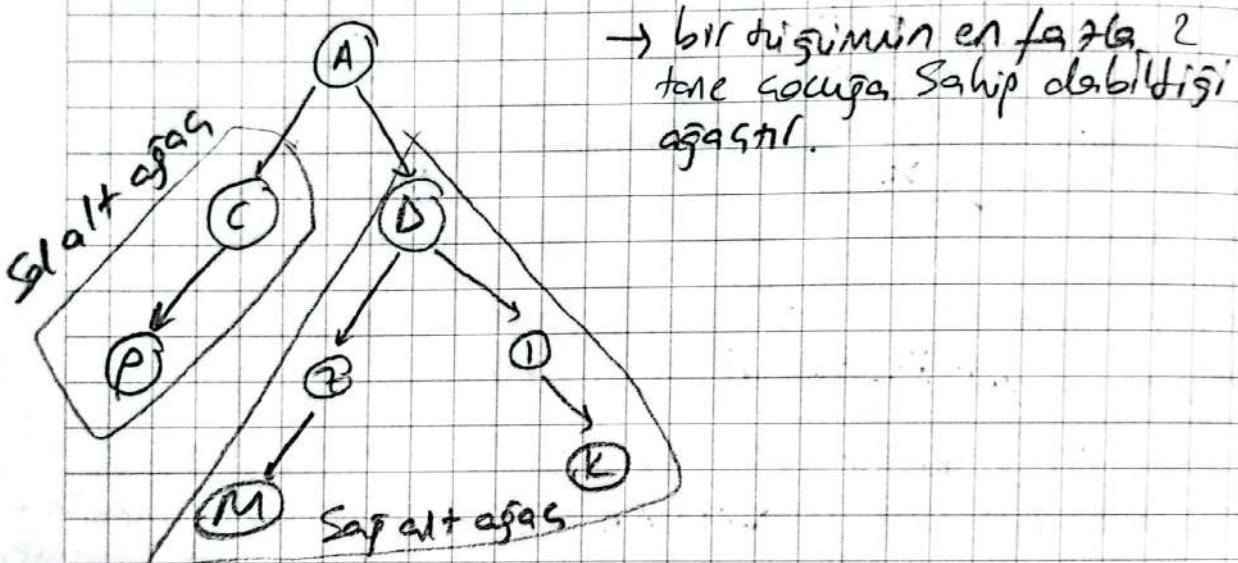
önemlidir. Bu veriler çalıştırılırken başka programlar gereklilik duyarlıdır. Bunun için path (yol) takımlaması yapılmalıdır.



NOT: N tane dugumden oluşan ağacın kesiş N-1 dir.

NOT: Ağactaki her dugum arasında en fazla 1 yol olabilir.

İkili Ağac (Binary Tree)



• Derinlik = 0 : $N=1 = 2^0$ = dugum

• Derinlik = 1 : $N=2$ ve 3 dugum = 2^1 ve $2^{1+1}-1$ dugum
D derinliginde $N=2^d$ ve $2^{d+1}-1$ dugum (tam bir ikili ağac)

NOT: Listenin en kötü durumu sıralı bir şekilde veriler geldiği zaman, o zaman ağac listeye deger ve cismi $O(n)$ dir.

En iyi durumda : $\log_2 N$ te calisir.

sol	veri	sag
-----	------	-----

```
public class IkitiAgacDugumu{
```

```
    public IkitiAgacDugumu sol;
```

```
    public int veri;
```

```
    public IkitiAgacDugumu sag;
```

```
    IkitiAgacDugumu DugumOlustur (int veri){
```

```
        IkitiAgacDugumu dugum = new IkitiAgacDugumu();
```

```
        dugum.veri = veri;
```

```
        dugum.sol = null;
```

```
        dugum.sag = null;
```

```
        return dugum;
```

```
}
```

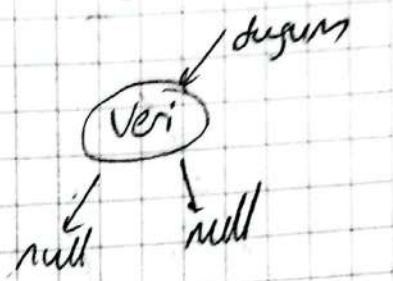
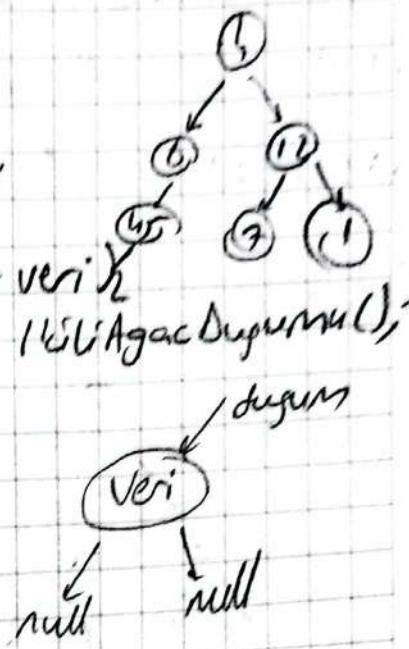
```
psvm(){
```

```
    kkt = DugumOlustur(4);
```

```
    kkt.sol = DugumOlustur(6);
```

```
    kkt.sag = DugumOlustur(12);
```

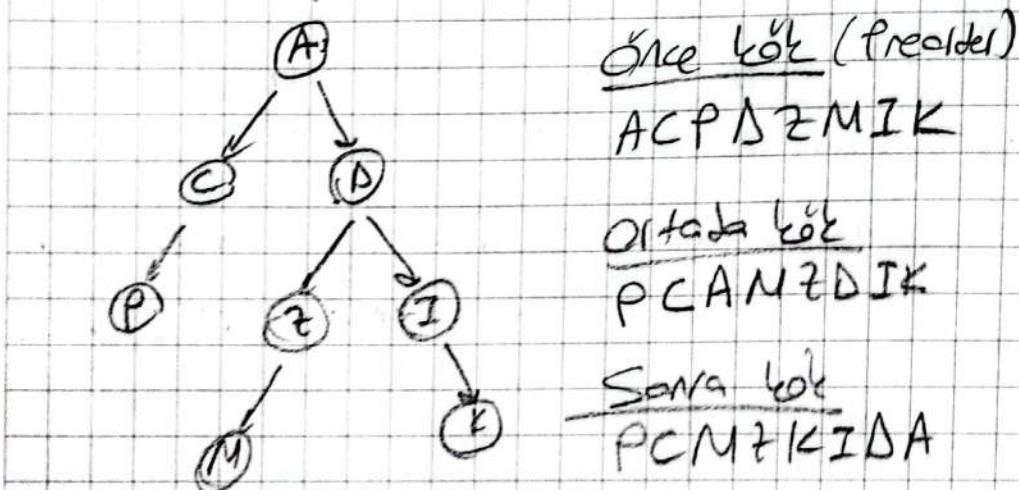
```
}
```



Ödev: Kestegeli sirilen 10 ile 100 arasındaki sayıları (toplam 10 adet) kendi oluşturacağınız ağac yerleştirin. Daha sonra bu ağac içerisinde preorder, inorder, postorder gezinme yöntemlerinde sayı dizilerini veriniz.

Illi Ağac Üzerinde Değerlendirme

- 1) Önce-kök (Preorder): kök, sol, sağ
- 2) Ortada-kök (Inorder): sol, kök, sağ
- 3) Sonra-kök (Postorder): Sol, sağ, kök



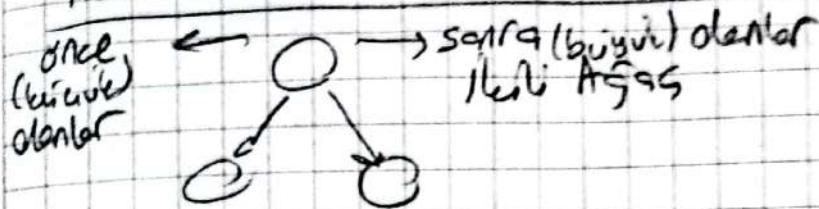
Once kök (İlli Ağac Dugumu kodu){

```
if(kök==null) return;  
cout(kök.veri + " ");  
Once kök(kök.sol);  
Once kök(kök.sag);}
```

Ortada kök (İlli Ağac Dugumu kodu){

```
if(kök==null) return;
```

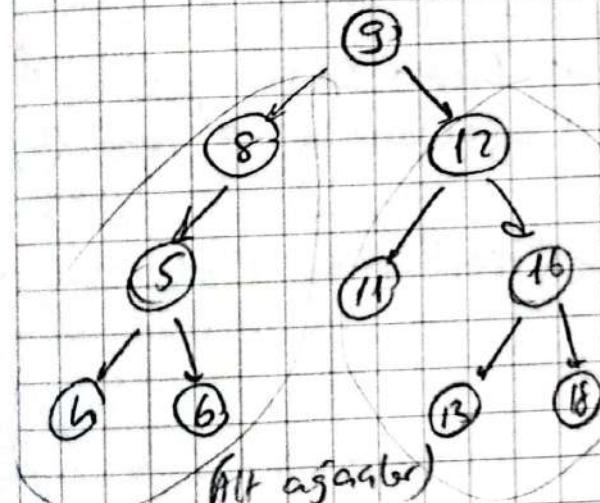
İkili Arama Ağacı (Binary Search Tree) BST



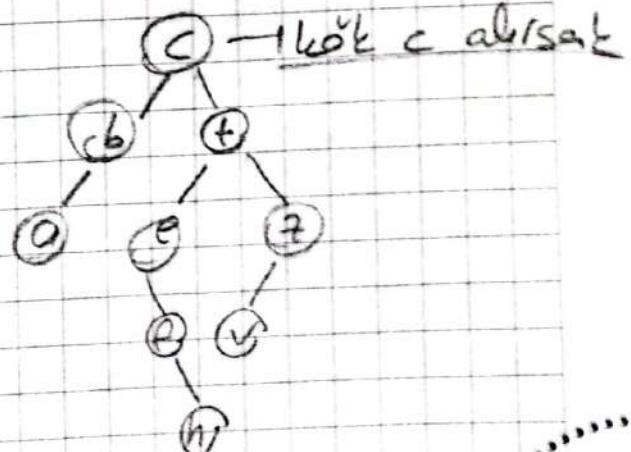
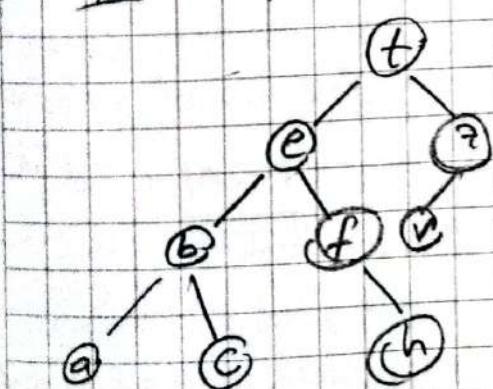
NOT: kök dizininin sağı küçüklere sola, büyükler
sağa yerleştirilir (sayılar için)

→ Karakterler (Stringler için) sözlükte önce gelenler sağa
sonra gelenler sağa yerleştirilir. (Ascii'ye göre)

ÖR) 8, 5, 12, 16, 18, 11, 4, 6, 13 dt (karakterlerin
genelde ilk sayı kök kabul edilir. dt gelis sırası
(i=0) ağacın soldan
başırılar.



ÖR) e, b, f, k, a, h, r, v ikili arama ağacına yerleştirin



DODI

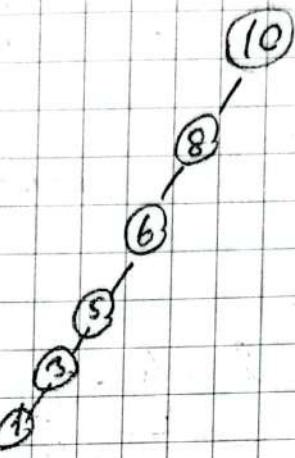
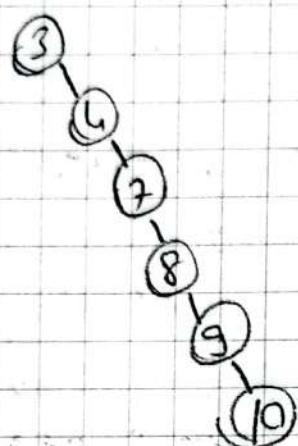
Dengeli Ağacı: en sondaki elemanı ilesmektir ve
en kusa yolu katetimde.
sağ ve sol yolların farklı maks 1 olmalıdır.
bu da almost ideal durum.

Özel Durum 1

(Dizi sıralı
gelirse)

3, 6, 7, 8, 9, 10 ($K \rightarrow B$)

10, 8, 6, 5, 3, 1 ($B \rightarrow K$)



→ bağlı liste'ye dönüştür
→ en kötü durum.

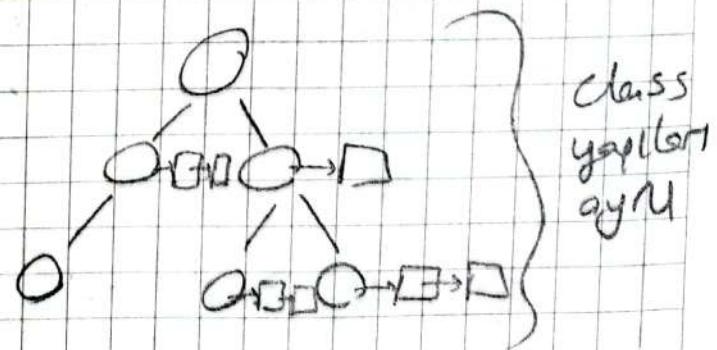
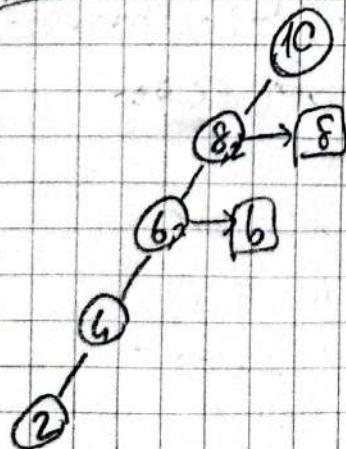
Özel Durum 2: Aynı elementten birden fazla gelince
bir yöntem vardır. (ilk önce orama ağacında ve heaplerde)

1) Sayas tutmak

2) Biriden fazla gelen düğümlerden itibaren liste
glüstürmek

→ class yapılan aynı

10, 8, 6, 4, 3, 8, 6



class
yapılan
aynı

Sınavda: Ağaca eklemme (AVL) + (ilhan Hoca)

Heap (Yığın) Ağacı ve Uygulamalar

• Ağacın dengeli olacak

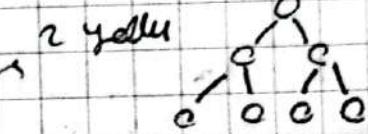
• Aile çocuklarından büyük olacak, ata aileden

Max, Min (Heap)

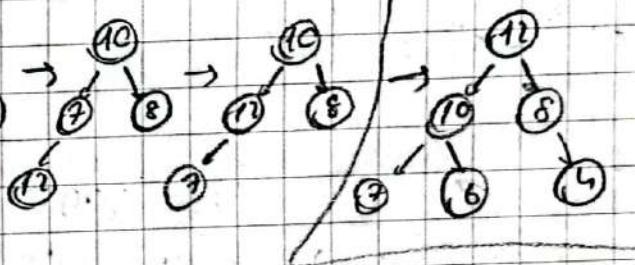
7, 8, 10, 12, 6, 5

$$\log_2 6 = 3$$

seviye



• en üsttaki değer
en büyük değer var
dizinin



Heap (Yığın):

- Yığın ağacı olarak geçer.

- İki türü vardır: • max
• min yığın ağacı

- Max yığın ağacında her zaman üst seviyedeki düğümleinin değeri bir alttaki değerlere göre daha büyükür.
- Min heap ağacında ise üsttaki düğümleinin değeri daha küçükür.
- Alt ağacın değerleri de bu işlemeye bağlıdır.
- Sıralama algoritmalarında yaygın bir şekilde kullanılır.

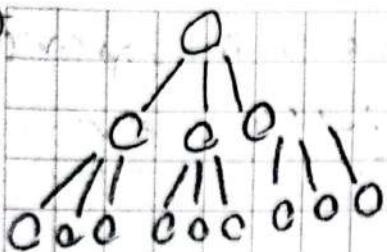
• 2, 3 veya 4 dallı olarak gerçekleştirilebilir.

• heap ağacı kesinlikle tamlı bir ağactır.

• ideal olan ağacın tamlı ve dallı olması

2 yıldan daha fazla olması, daha fazla veren

do



\log_2

Souvielii
olacore

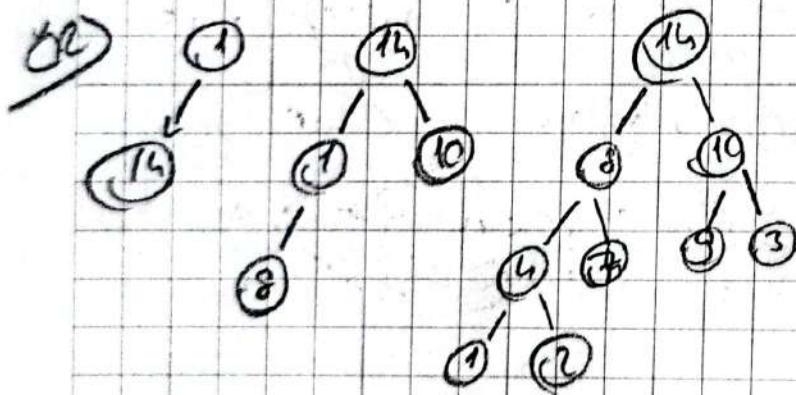
A simple line drawing of three stylized human figures standing side-by-side. The figures have circular heads and thin bodies. The figure on the left has a small circle at the bottom of its body. The figure in the middle has two small circles at the bottom of its body. The figure on the right has a single small circle at the bottom of its body.

$$\log_4$$

İkili arama ağacındaki gibi soldakiler, saptanergusle
değildir.

• bir çalışma offline ise elinde bütün veriler var. online ise veriler sürekli gelmeye devam ettiğidir

de bir again tengeli olması dolu olduğuna onlarla
gelmeyip & tamamen dolu öğrenciler tengelidir.



16 8 10 6 7 9 3 2 1

Yığın İslemleri

Etwas : Objekt N)

Silme : $O(\log N)$

Deger degistirme: $O(\log N)$

Arama işlemi $O(N)$

^{2.yerde}
* M adet sıralı diziyi heap kullanarak Birleştirme
islemi için minimum heap ağacı kullanmak esastır.

* Algoritması $O(N \log M)$ dir.

birincinin 1. n. elemen
elemanı. 2. n. elemen
sayısı

B Trees

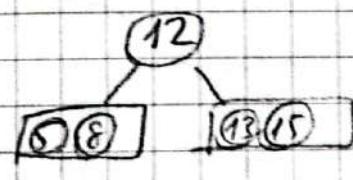
* Sade yolu ağaçlardır.

* Butun yapraklar aynı seviyededir.

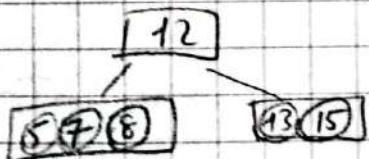
* Özellikle büyük verilerin işlenmesinde ve tutulmasında kullanılır.

Kök (root), en az iki tane yaprak olmayan node's sahiptir.

* her yaprak düğümü $k-1$ anahtara sahiptir.



Sade Yolu Ağaçları B-trees
1) Yerlestiristecde yaprak bos ise
7' nin eklenmesi



NOT: B ağaçları heap ağacının özel bir türüdür.
Kökte en az iki tane yaprak olmayan düğümü
barındırmalıdır.

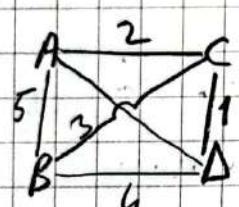
NOT: (32.slayt) Güzel bir B ağacı oluşturmak için
kökte en az iki tane düğümle çalışılması istenir

(33.slayt) NOT: B ağaçları istekli düğüme bağlı olarak
kontrol içinde kucukluk - büyüklik degerine uygun
ağaçlardır.

Graf ve Algoritmalar: Graf, yönü ya da yönüs selüle çizilen iki ya da daha fazla düğümü稳定的 gelen yapılardır. Türkçeye, çizge olarak geçirmistir.

Min. maliyet, Min. harcaması, Min. yol alma problemlerinin çözümünde kullanılır.

Elektrik devrelerinde, devre çözümünde, bilgisayar teknolojisinde, navigasyon sistemlerinde, gezgin satıcı problemleri çözümde... grafler ve algoritmalar kullanılır. Cihazların birbirileriyle iletişiminde grafler kullanılır.



$G \rightarrow$ ile gösterilir

$$G = \{E, V\}$$

Edge
(kenar)

Vertex
(köşe, düğüm)

$$G = \{E, V, W\}$$

weight

$$V = \{A, B, C, D\}$$

* Spanning tree de ki

$$\text{kenar sayısı} = V - 1$$

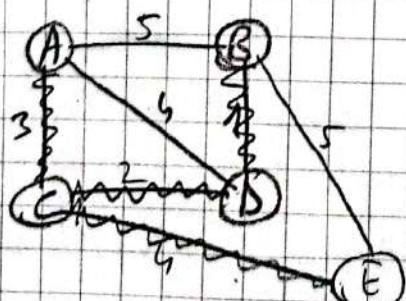
$$W = \{5, 2, 1, 3, 4\}$$

graph'taki
köşe sayısı

$$E = \{AB, AC, CD, BC, BD\}$$

Minimum Dolasma Ağacı (Minimum Spanning Tree)

1) Prim Algoritması: Min. dolasma ağacı algoritmasıdır. Bir graftaki bütün düğümlerin min. maliyetle, min. ağırlıkta dolasmasını sağlayır algoritmasıdır.



* Prim algoritması herhangi bir düğümden başlayabilir.

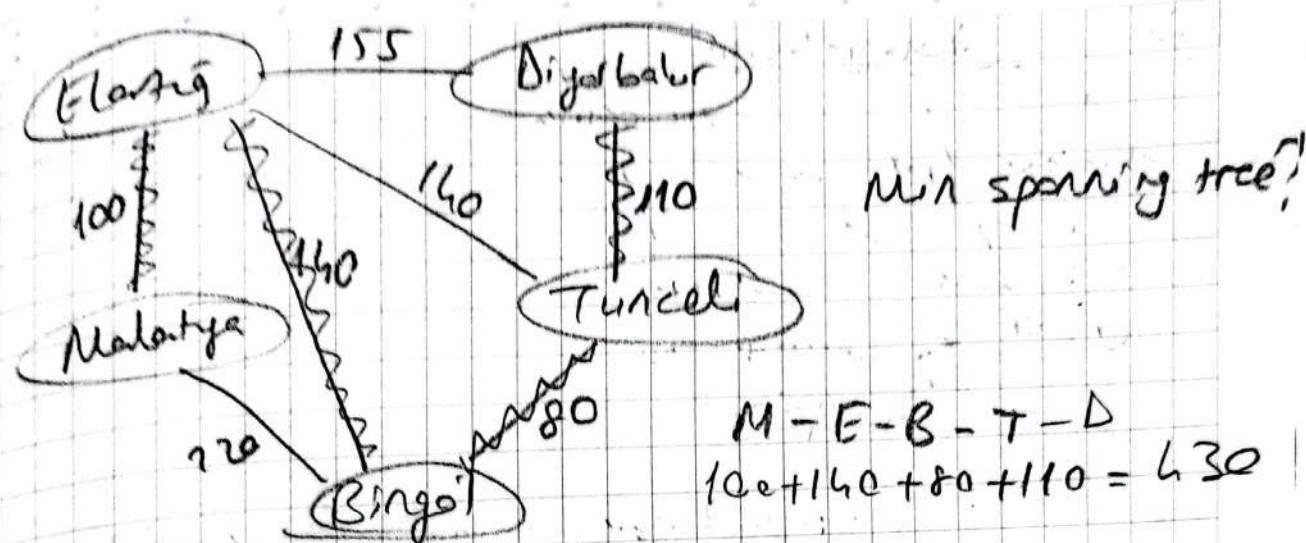
ACDBE

$$3 + 2 + 1 + 4 = 10 \text{ min.}$$

spanning değeri

* Esit olan düğümlerden herhangi birini seçebiliriz.

* spanning tree is a sub graph of a graph
(Spanning: yes/len)



2) Kruskal Algoritması: Min. spanning tree algorithması.

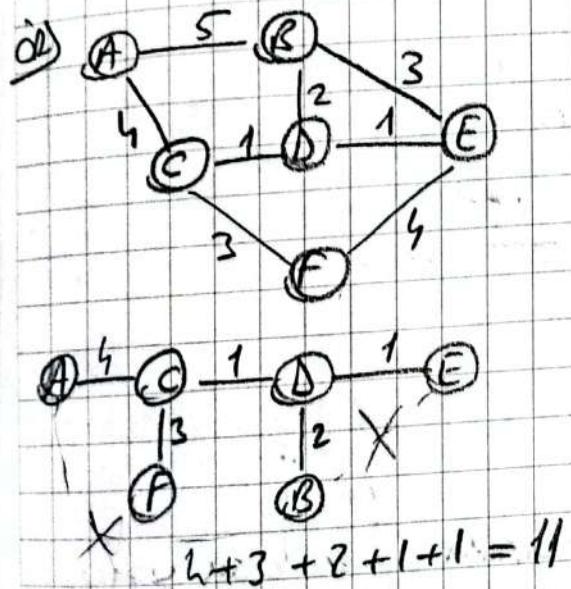
↳ greedy approach

↳ her sonan minimum konsantresi faktat devre oluşturuyorsa alır. o konsantresi sil.

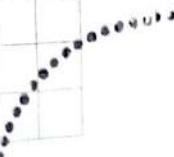
↳ prim algoritmasıyle aynı sonucu verirler

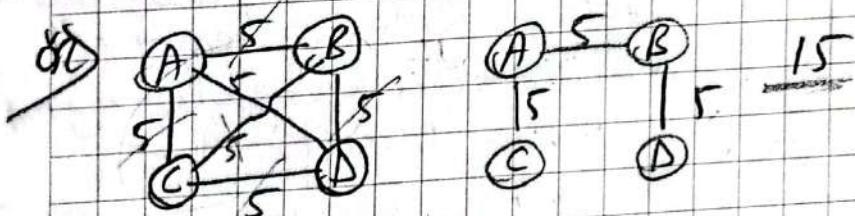
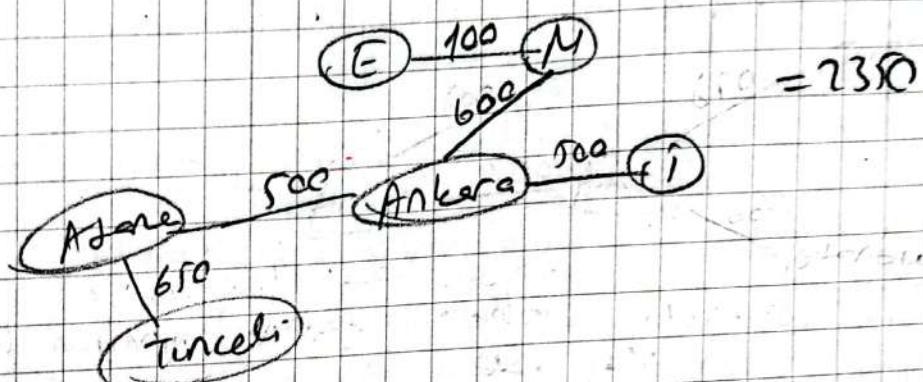
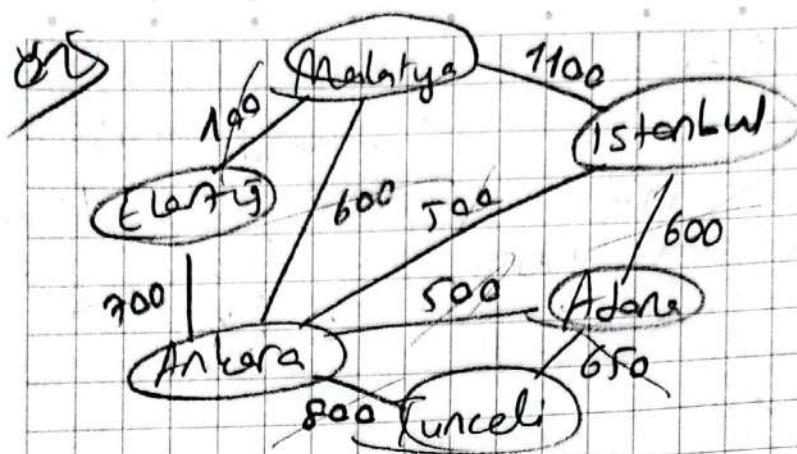
↳ zaman karmaşıklığı: $O(n^3)$

↳ min heap yapısıyla gerçekleştirilirse: $O(n \log n)$



82)

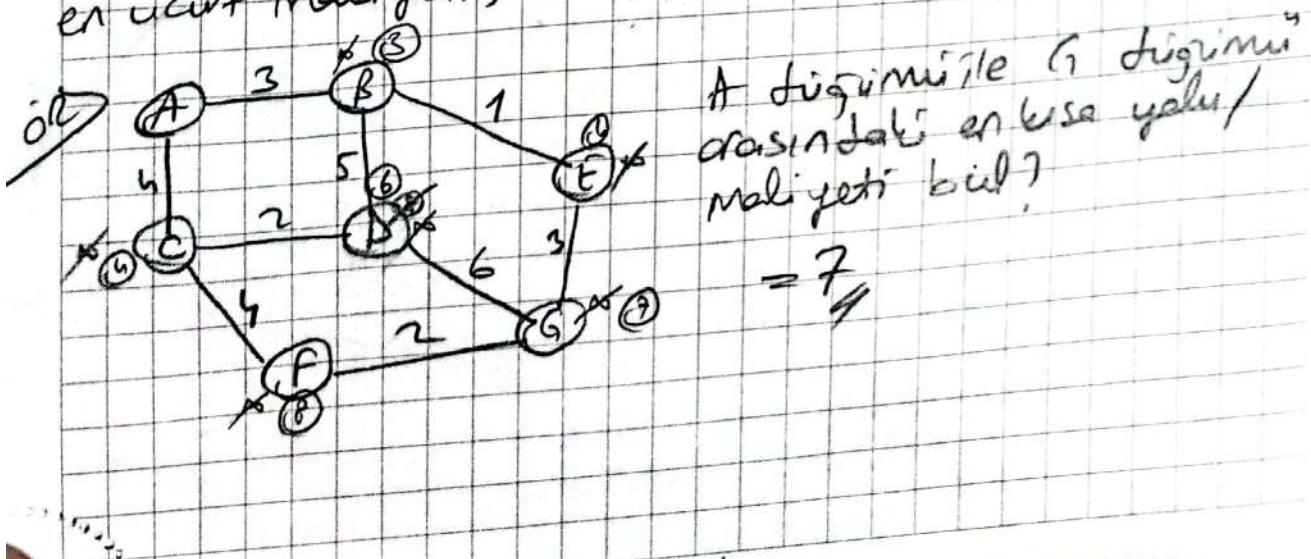


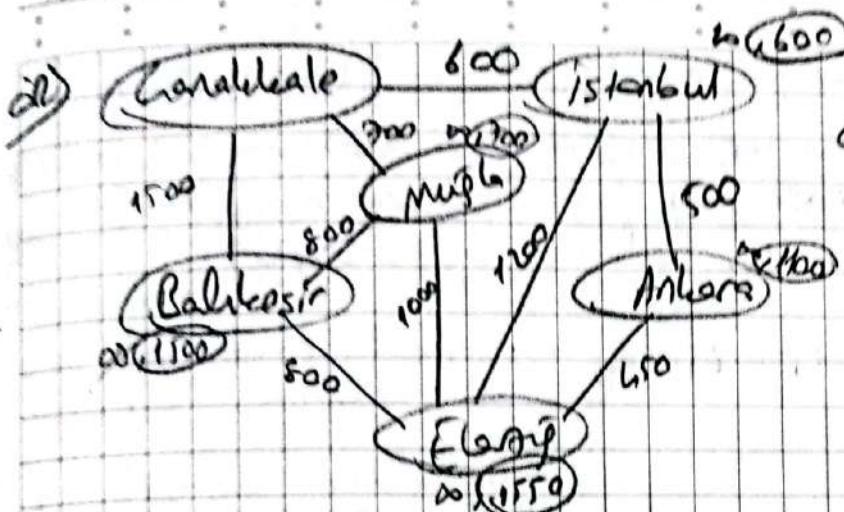


Shortest Path Algoritması (En kısa yel algoritması)

Dijkstra Algoritması

→ Bir graffaki ikinci düğüm arasındaki en kısa yolu, en ucuz maliyeti, en kısa süreyi onluk olarak hesaplar.





Catalca - Elazığ
arası en kısa yolda?

1500 en kısa
yol

Arama Algoritmaları

• Arama algoritmalarının amacı tek boyutlu ya da
daha yükselt boyutlu veriler içinde aranan bir
elemanın var olup olmadığıdır; var ise kaçinci
eleman olduğuğun bulunmasıdır.

İki tane temel yöntemini tanıtmaktır:

1) Lineer arama (doğrusal) $\rightarrow O(N)$

ile boyutlu dizi $O(n^2)$)
3 " " " $O(n^3)$

2) İkili arama

$$A = \{3, 1, -2, 6, 7, 5\}$$

public int lineer (int[] A, int aranan) {

for (int i = 0; i < A.length; i++) {
if (A[i] == aranan) return i;

return -1;

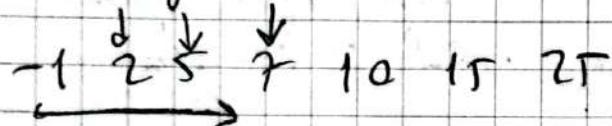
}

Tümde bolme varsa $\log_2 N$

Cuda Stream?

İkili Arama (Binary Search)

→ ikili aramada sayılar ya da dizilerin sıralı olması gereklidir. Sıralı denildiği zaman bulgısının $k \rightarrow \beta$ ya da sıralı tek bir haliyle düşünürlür.



Araçan! ②

```
public static int binarySearch(int[] a, int target){  
    int min = 0;  
    int max = a.length - 1;  
    while (min <= max) {  
        int mid = (min + max) / 2;  
        if (a[mid] < target) {  
            min = mid + 1;  
        } else if (a[mid] > target) {  
            max = mid - 1;  
        } else {  
            return mid; // target found  
        }  
    }  
    return -(min + 1); // target not found
```

NOT: elastic search → çok hızlı

Sıralama Algoritmaları

Rastgele verimsiz olan şartların küçükten büyüğe, ya da $B \rightarrow K$ e doğru belirlenmesidir.

→ Literatürde 27'ye yakın sıralama algoritması var. Bunların bazıları sequential mantıkla, bazıları paralel çalışır.

A) Artisal Çalışmalar

B) Divide and Conquer (Böl ve Yönet algoritması) (Recursive çalışması daha uygundur)

NC: Genelde istatistiksel amaçlı olarak kullanılır.

Artisal Çalışmalar

- Serum sıralama Algo (selection sort)
- araya eklemeye algo (insertion sort)
- kabarcık sıralama algo (bubble sort)

→ merge sort, quick sort ise rekursif şekilde çalışabilir.

→ Sıralama algoritmaları en kötü $O(n^2)$
en iyi: $O(n \log n)$

Selection Sort

```
template < class Item >
void selectionSort (Item a[], int n) {
    for (int i = 0; i < n - 1; i++) {
        int min = i;
        for (int j = i + 1; j < n; j++) {
            if (a[j] < a[min])
```

Not: $O(n^2)$ de çalışır.

* Selection Sort algoritması offline çalışır.

Yeni verilerin bulunması gereklidir.

Insertion Sort (Araya ekleme algo)

→ Bu algoritma hem online hem offline çalışır

* → Online çalışmayı desteklediği için gerçek zamanlı uygulamalar da kullanılabilir.

Online

$t=0$	5
$t=1$	5, 2
$t=2$	6 2 4 5
$t=3$	-6 F 6 2 4 5

* online işlemler mikro-
işlemci yapısına da
uygundur.

```

template < class Item >
void insertionSort( Item a[], int n) {
    for( int i=1 ; i<n ; i++ ) {
        Item tmp = a[i]; // the element to be
                           inserted
        int j;
        for( j=i ; j>0 && tmp < a[j-1] ; j-- )
            a[j] = a[j-1]; // shift elements
        a[j] = tmp; // insert
    }
}

```

Bubble Sort

23	78	45	8	32	56	
23	45	78	8	32	56	$O(n^2)$
23	45	8	78	32	56	
23	45	8	32	78	56	
23	45	8	32	56	<u>78</u>	
23	45	8	32	<u>56</u>	78	
23	8	45	32	56	<u>78</u>	
23	8	37	<u>45</u>	<u>56</u>	<u>78</u>	

```

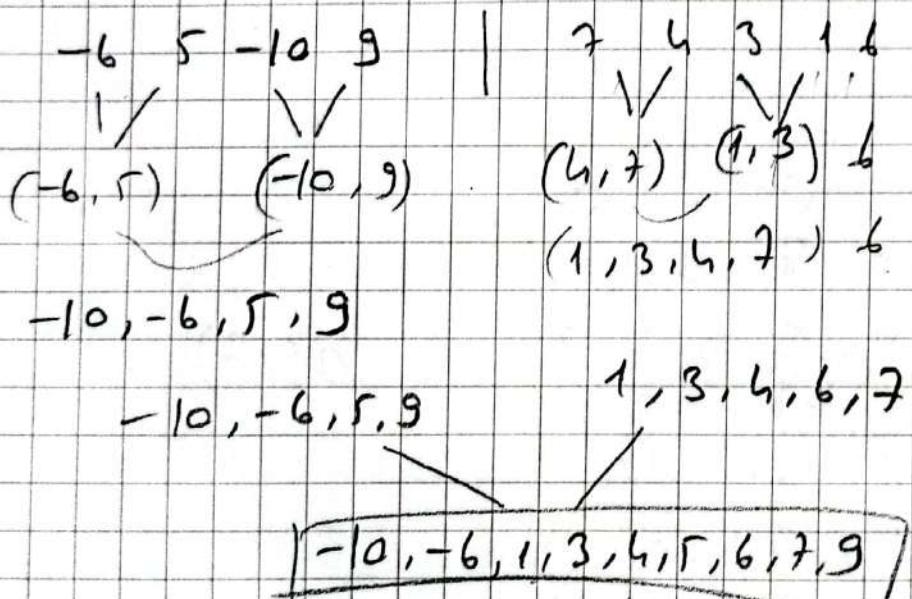
template < class Item >
void bubbleSort( Item a[], int n) {
    bool sorted = false;
    int last = n-1;
    for( int i=0 ; (i<last) && !sorted ; i++ ) {
        sorted = true;
        for( int j=last ; j>i ; j-- ) {
            if( a[j]<a[j-1]

```

Merge Sort (Ekleme Algoritması)

- Bö ve yonet algoritmalarından bir tanesidir.
- Rekursif analiz.
- her bir kusim ayrik bir sekilde rekursif olarak siralenabilir. Daha sonra bu kusimlar billestirilmek suretiyle sirali dizisi elde edilir.

Q) $-6 \ 5 \ -10 \ 9 \ | \ 7 \ 4 \ 3 \ 1 \ 6$ dizisinin merge sort algo. ile siralayın.



- * her asonda ikili gruplar siralanacak
 - * 3'li olsa da fikr etmez, ikili yapacaptı.
- $O(n \log n)$

Quick Sort Algoritması (Hızlı Sıralama)

Calışması:

• bir tane pivot değeri seçilir.

↳ 1) rastgele bir değer seçilebilir.

2) dizinin ortasındaki element seçilebilir

3) dizinin ortalama değeri bulunur. Bu değerde baslanır (dizide olmazsa)

4) dizide olmayan bir değerle başlanabilir ama aynı sayıları içindeki bir değer seçilir.

• Algoritmalar içinde en hızlıdır

$O(n \log n)$

• Quick sort algoritması dizisi sıralı gelirse en kötü çalışır $O(n^2)$

• pivotu her zaman da relativif olarak yeri bulur.

5	3	2	-1	6	
-1	2	5	3	6	→ 2.pivot
1	2	3	5	6	